GRASP: Generic Reasoning And SPARQL Generation across Knowledge Graphs - Demo System

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

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

Abstract

GRASP is the first approach for SPARQL-based question answering that, in principle, works for arbitrary given RDF knowledge graphs *zero-shot*, that is, without prior training or information on the graph. In this work, we present and describe a prototypical demo system that implements the GRASP approach. The system also supports general question answering and follow-up questions. We extend the evaluation of the associated research paper by experiments on the IMDb knowledge graph and the TEXT2SPARQL challenge.

Keywords

Question Answering, SPARQL, Knowledge Graphs

1. Introduction

GRASP is an approach for answering questions on arbitrary given knowledge graphs without the need for training or graph-specific information [1]. GRASP accepts an arbitrary question in natural language and tries to find a SPARQL query on the given knowledge graphs, using a fixed set of pre-defined functions. The function calls are controlled by a large language model (LLM) and serve to interactively explore the knowledge graphs, much like a human would when confronted with the same task. In [1], this approach was shown to give good results for a variety of knowledge graphs, including popular ones like Wikidata [2], Freebase [3], and DBLP [4].

The goal of this work is to build a complete system based on this approach. The system is composed of the LLM-based agent (see Section 3.1), knowledge graph indices (see Section 3.2), and a web application that allows users to interact with the agent. Fig. 1 provides an overview of the system.

Contributions

1. We have built a complete system based on the GRASP approach. We provide a detailed description of the architecture and the core components (see Section 3).
2. Beyond the core SPARQL question answering functionality, the system supports multi-turn follow-up questions, as well as general-purpose question answering. These modes can be freely mixed.
3. We extend the evaluation of [1] by experiments on IMDb [5] and the TEXT2SPARQL challenge [6].
4. We provide all code, indices, and documentation at github.com/ad-freiburg/grasp. Users can easily use GRASP via a CLI, setup their own server, or visit the public demo at grasp.cs.uni-freiburg.de.

2. Related Work

The SPARQL QA problem fits within the broader domain of knowledge graph question answering, which can be divided into three categories. The first category includes methods that are fine-tuned for a specific benchmark and knowledge graph [7, 8, 9]. These methods often achieve strong results by being able to adapt to patterns in the benchmark and knowledge graph. The second category includes

ISWC 2025 Companion Volume, November 2–6, 2025, Nara, Japan

*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)

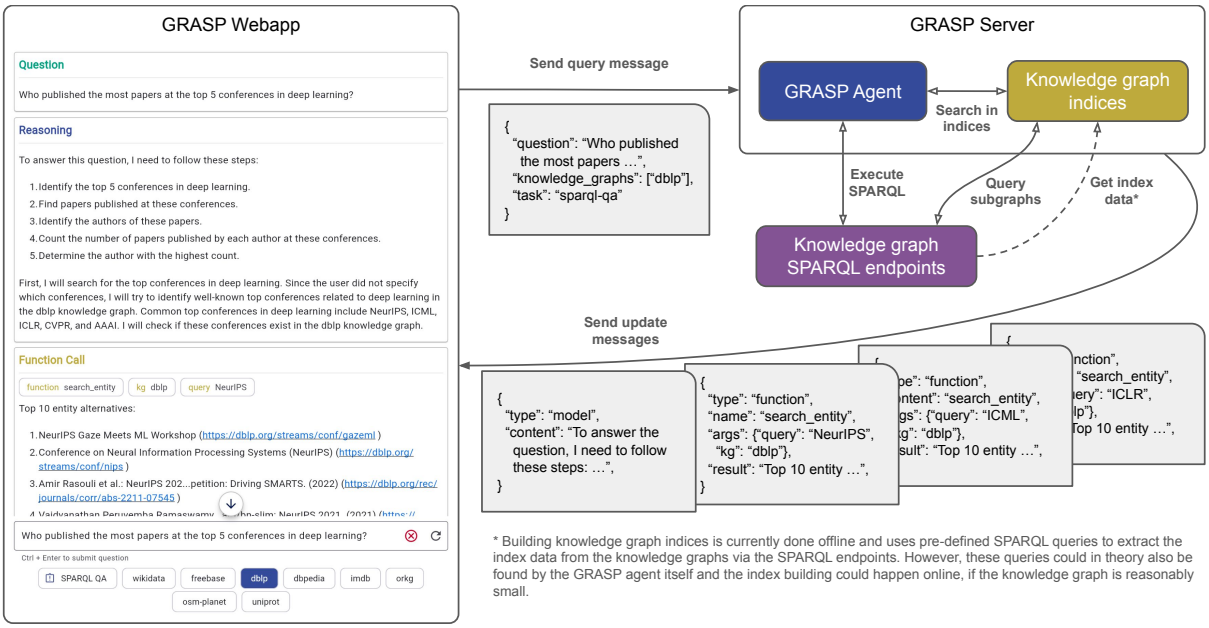

Figure 1: The GRASP client-server setup allows the user to invoke the GRASP agent via an intuitive web application. The user initiates the process by sending a query message containing the question, knowledge graph selection, and task name to the server. The server starts a GRASP agent that begins exploring and querying the specified knowledge graphs to answer the question according to the task. Progress updates are sent back to the web application after each reasoning step or function call of the agent and displayed there upon arrival.

methods that do not require fine-tuning but instead use in-context learning with information that is specific for the benchmark or knowledge graph, such as exemplary question-SPARQL pairs [10, 11, 12]. The third category covers methods which function without any of the above, but rather explore the knowledge graph on the go [1, 13, 14, 15]. For a more detailed account of related work, we refer to [1].

Only one of the latter methods, SPINACH [13], provides a publicly available demo of their approach at spinach.genie.stanford.edu, which uses a chat-like interface similar to ours.[1] But like its underlying approach, this system is limited to the Wikidata knowledge graph.

3. System

In the following, we describe the main components of our system: an LLM-based agent that controls the function calls (Section 3.1), the indices used by the agent to search the knowledge graphs (Section 3.2), and the configuration allowing the user to adapt the system to their needs (Section 3.3).

The GRASP CLI is the main tool for working with the GRASP system. It is used to prepare knowledge graphs, build indices, start the GRASP server, run the GRASP agent in a headless fashion, and more. See Fig. 2 for an overview. Most users will use GRASP in a client-server setup. For that, we also provide a compatible web application. See Fig. 1 for an overview.

3.1. GRASP Agent

In a nutshell, the GRASP agent works as follows: starting from the user's question augmented by a generic instruction prompt, it enters a loop, querying the knowledge graph exploratively until it finds a SPARQL query that produces the desired answer. It reasons about previous query results to determine the next query or whether the final answer has been found. The queries are realized via *function calling*.

Specifically, the agent is provided with a fixed set of functions, each with a unique name and a

[1] See also www.wikidata.org/wiki/User:SpinachBot

```
# Run GRASP agent on a question
echo "Name 10 german race car drivers" |
grasp run <config>
# Run GRASP agent on multiple questions
cat questions.jsonl | grasp file <config>
# Start the GRASP server
grasp serve <config>
```

```
# Get search index data for a knowledge graph
grasp data <kg> [--endpoint <endpoint>]
# Build search indices for a knowledge graph
grasp index <kg>
# Build an example index from question-sparql-pairs
grasp examples examples.jsonl examples.index
# Evaluate GRASP's predictions on a benchmark
grasp evaluate benchmark.jsonl pred.jsonl <endpoint>
```

Figure 2: The GRASP CLI: On the left, we show how to run GRASP on single questions, question files, and as a server. On the right, we show how one can prepare knowledge graphs and SPARQL QA examples for the use with GRASP, and how to evaluate GRASP's predictions on a benchmark. `<config>` refers to the path of a configuration file (see Section 3.3), `<kg>` to the name of a knowledge graph, and `<endpoint>` to a SPARQL endpoint. All commands have additional optional flags not shown here for brevity.

Table 1
Statistics about the average number of steps, function calls, and time it took GRASP with GPT-4.1 to answer questions on the main benchmarks used in [1]. We also show how function calls are distributed among all functions (ignoring ANS and CAN). F_1-scores are taken from [1].

			Average			Distribution (%)					
KG	**Benchmark**	**F_1-score**	Steps	Fn calls	Time	EXE	LST	SEN	SPR	SPE	SOP
Freebase	CWQ [21]	44.2	19.5	14.0	37.5 s	8	55	14	5	18	0
	WQSP [22]	52.1	16.3	11.5	29.8 s	11	58	12	4	15	0
Wikidata	QALD-10 [23]	72.5	10.7	7.0	22.5 s	23	25	27	15	9	0
	QALD-7 [24]	79.4	10.1	6.2	18.2 s	28	17	28	22	5	1
	SPINACH [13]	40.8	15.9	10.9	43.8 s	20	29	22	25	4	0
	WWQ [25]	75.3	10.3	6.4	21.4 s	25	25	24	14	12	1

description in natural language of its purpose and its parameters.[2] The functions are: EXE (execute an arbitrary SPARQL query), LST (list triples with given constraints), SEN (search for entities matching a given query string), SPR (search for properties matching a given query string), SPE (search for properties of a given entity), SOP (search for objects of a given property), ANS (answer and stop), CAN (cancel and stop). If few-shot examples are available, one of the functions FSE (find similar examples) or FEX (find random examples) is provided as well. See [1] for a more detailed description of each of these functions.

For this paradigm to work properly, we rely on the underlying model being trained to support zero-shot function calling, which is true for nearly all recent closed-source models like GPT-4.1 by OpenAI [16] or Gemini 2.5 by Google [17], as well as for many recent open-source models like Qwen2.5 [18] or Qwen3 [19]. The latter can be easily self-hosted via vLLM [20]. Wherever supported by the model provider, we use constrained decoding to force the model to output valid function calls that are guaranteed to follow the available function signatures for increased reliability.

Table 1 provides F_1-scores and statistics on six benchmarks. For CWQ, WQSP, and SPINACH, GRASP achieves a comparatively low F_1-score and, on average, uses more steps, function calls, and time. We suspect that this is due to the harder questions, in particular on SPINACH. For CWQ and WQSP, we observe more LST and fewer EXE calls than on the other benchmarks. We assume that this is due to the older Freebase knowledge graph being less familiar to the underlying LLM, which thus requires more exploration and verification steps (which GRASP typically realizes via LST). We also find that the SOP function is almost never used; it could therefore probably be removed without loss of performance.

Tasks

Not all questions are answerable by or via a SPARQL query. We have therefore implemented an extension that allows the user to dynamically switch to the task of general question answering over

[2]The implementation of the functions is fixed and part of the GRASP system. When calling a function, the model provides the function name and parameter values, and receives back the results in text format from our implementation.

knowledge graphs by adapting the GRASP instruction and ANS and CAN functions respectively. For this task, the final output is not a SPARQL query, but arbitrary natural language text. For example, this is useful for a question like "Write a Python script to download all Wikipedia articles about dog breeds", which can be answered by first finding a SPARQL query to retrieve the article URLs, and then using this SPARQL query in a Python program.

Follow-up question answering

In [1], the GRASP agent is used to answer questions by finding a corresponding SPARQL query in a single uninterrupted interactive process between model and knowledge graphs. In practice, one would like to have multi-turn conversations and ask follow-up questions, potentially switching tasks and the underlying knowledge graphs at each subsequent question. We implement this by first determining the GRASP instruction for the current task and knowledge graph selection, then adding all previous questions and reasoning or function call steps unchanged, and finally asking the follow-up question. The web application supports this use case by sending an additional *past* field containing the full interaction history on follow-up questions.

3.2. Search indices

Besides the agent, the search indices are the second integral part of the overall GRASP approach. Currently, the GRASP system supports two types of search indices: prefix-keyword indices (PFX) for prefix-sensitive keyword search, and similarity indices (SIM) for vector-based similarity search. We refer to [1] for a more detailed explanation of both index types.

The indices enable search queries that are either inefficient in SPARQL (in the case of prefix-keyword search) or not supported by SPARQL (in the case of vector-based similarity search). We make the search indices accessible to the GRASP agent via easy-to-use search functions (see Section 3.1), which are purpose-built for the most common types of searches a human expert performs while writing SPARQL queries. It is shown in [1] that these search functions (implemented using the mentioned indices) significantly boost the overall system performance (compared to when only EXE is provided).

For entities, we build a PFX by default, because it requires less disk space and RAM, and is faster to query than a SIM. Besides, SIM does not give significantly better results because keyword search works well on entities. Given a PFX query, we calculate a score between each entity and the query as the weighted sum of the number of exact and prefix keyword matches minus a weighted sum of the number of unmatched query and entity keywords. If there is neither an exact nor a prefix keyword match, the entity is excluded entirely. The entities with the k highest scores are then returned as search results, where k is a GRASP configuration option (see `search_top_k:` in Fig. 3).

For properties, keyword queries often miss relevant search results. For example, searching for "born in" should also match "place of birth", but does not when using a PFX. We therefore build a SIM for properties by default. Since knowledge graphs typically have only few properties, the higher disk space and RAM consumption of SIM compared to PFX does not matter. Given a search query, we compute its vector embedding (using Qwen/Qwen3-Embedding-0.6B [26] by default), compute the cosine-similarity to all pre-computed property embeddings and return the top k properties with the highest similarity as search results. Note that a GPU is required to run a SIM efficiently in practice.

Both PFX and SIM support specifying a subset of items to restrict the search to. Together with a SPARQL endpoint we use this functionality to implement the advanced search functions of GRASP. For example, SPE allows searching for properties of a given entity. For that, we first send a SPARQL query to the endpoint to retrieve all potential properties for the given entity, restrict the corresponding property index to these properties, and execute the search query over that restricted index.

Table 2 provides statistics for the search indices of eight knowledge graphs. Starting the GRASP server with all these indices takes less than 20 s, and uses ≈ 20 GB of RAM and ≈ 3 GB of GPU memory, measured on a machine with an AMD Ryzen 9 7950X CPU, an NVIDIA GeForce RTX 4090 GPU, and 4 × 4 TB NVMe SSD.

Table 2
Statistics about our search indices for major knowledge graphs, summed over entities and properties. *Size* refers to number of indexed items, *Disk* to the total size on disk, and *Build* to the total build time.

KG	Size	Disk	Build
DBLP [4]	141 M	22 GB	271 s
DBpedia [27]	19 M	3 GB	74 s
Freebase [3]	23 M	3 GB	62 s
IMDb [5]	26 M	3 GB	34 s
ORKG [28]	0.5 M	0.1 GB	14 s
OSM [29]	97 M	11 GB	243 s
UniProt [30]	4 M	0.6 GB	11 s
Wikidata [2]	84 M	14 GB	193 s

Table 3
F_1-score of GRASP with GPT-4.1 on our own IMDb benchmark and the domain-specific knowledge graph from the TEXT2SPARQL challenge representing a corporate setting. For TEXT2SPARQL, we evaluate all methods with both the evaluation metric of [1] and with the official TEXT2SPARQL evaluation tool at github.com/aksw/text2sparql-client. The first score refers to the former, the second to the latter.

Benchmark	Method	F_1-score
IMDb	GRASP	57.8
TEXT2SPARQL (corporate)	GRASP	77.5 / 50.2
	INFAI	61.1 / 47.9
	IIS-Q	54.2 / 44.5

```yaml
# Model setup
model: env(MODEL:openai/gpt-4.1)
model_endpoint: env(MODEL_ENDPOINT:null)

# Model inference
model_kwargs: env(MODEL_KWARGS:null)
temperature: env(TEMPERATURE:0.2)
top_p: env(TOP_P:0.8)
reasoning_effort: env(REASONING_EFFORT:null)

# System behavior
seed: env(SEED:null)
fn_set: env(FN_SET:search_extended)
feedback: env(FEEDBACK:false)
max_feedbacks: env(MAX_FEEDBACKS:2)
know_before_use: env(KNOW_BEFORE_USE:false)

# Function behavior
list_k: env(LIST_K:10)
search_top_k: env(SEARCH_TOP_K:10)
result_max_rows: env(RESULT_MAX_ROWS:10)
result_max_columns: env(RESULT_MAX_COLUMNS:10)

# Token, time, and message limits
max_completion_tokens: env(MAX_TOKENS:8192)
completion_timeout: env(TIMEOUT:120)
max_messages: env(MAX_MESSAGES:200)

# Few-shot examples (index specified per KG below)
force_examples: env(FORCE_EXAMPLES:null)
random_examples: env(RANDOM_EXAMPLES:false)
num_examples: env(NUM_EXAMPLES:3)

# Knowledge graphs
knowledge_graphs:
  - kg: env(KG:wikidata)
    # if endpoint is null, we fallback to
    # https://qlever.cs.uni-freiburg.de/api/<kg>
    endpoint: env(ENDPOINT:null)
    example_index: env(EXAMPLES:null)
    entities_type: env(ENTITIES_TYPE:null)
    properties_type: env(PROPERTIES_TYPE:null)
  # more knowledge graphs can be added here
  # - kg: dblp
  #   ...
```

Figure 3: Configuration options for GRASP in YAML format with sensible defaults for the GRASP agent based on GPT-4.1 over Wikidata. The user can either set an option directly in the YAML file or via environment variables using placeholders of the form `env(VAR_NAME:default)`. We show all options for completeness here.

3.3. Configuration

GRASP can be easily configured via a single configuration file in YAML format. See Fig. 3 for the file structure and configuration options. We provide sensible defaults for all configuration options, so the user often only needs to configure the particular knowledge graphs they want to use with GRASP. For example, a YAML config to run GRASP with Wikidata and Freebase can be as simple as `knowledge_graphs: [kg: "wikidata", kg: "freebase"]`. With the client-server setup, all configured knowledge graphs are automatically available in the web application, which itself requires no configuration; only the address and port of the GRASP server need to be set.

We briefly discuss the three most important configuration options for GRASP besides the model and knowledge graphs themselves:

1. Setting `feedback: true` corresponds to GRASP-F from [1] and allows the GRASP agent to reflect on and improve its own answers, which increases quality at the cost of longer runtimes. The

`max_feedbacks`: option sets the upper bound for the number of feedback loops per generation.

2. Setting `force_examples`: to a knowledge graph that specifies an example index via `example_index`: triggers a call of either the `FEX` or `FSE` function (depending on whether `random_examples`: is set to `true` or `false`) at the beginning of a generation. This enables few-shot learning in the style of the few-shot evaluations from [1].

3. Setting `know_before_use`: `true` tells GRASP to verify knowledge graph items before using them in `EXE` function calls. This is enforced by returning an error message rather than the query result if an `EXE` call uses items that were not present in any previous function call result. This mechanism avoids hallucinations of knowledge graph items, which we found to be a frequent problem with GRASP, in particular on knowledge graphs that are less familiar to the underlying LLM. For example, for the DBLP-QuAD [31] benchmark, without this setting, GRASP often uses incorrect properties without verification, like the seemingly more canonical *dblp:author* instead of the correct *dblp:authoredBy*. Consequently, this setting improves the F_1-score on this benchmark from 51.0 to 66.8.

4. Additional Evaluations

To further validate GRASP's zero-shot question answering capabilities, we extend the set of evaluated knowledge graphs and benchmarks from [1]. First, we build an own small benchmark for IMDb [5], the popular movie and series database, consisting of 15 questions. Second, we evaluate GRASP on the small, domain-specific knowledge graph representing a corporate setting from the TEXT2SPARQL challenge [6]. The corresponding challenge benchmark contains 50 questions and is "designed to test a model's ability to adapt to restricted and domain-focused data environments". On both benchmarks, GRASP achieves good results and even surpasses the best (INFAI) and second best (IIS-Q) entries from the TEXT2SPARQL challenge by a large margin. See Table 3 for full results.

5. Conclusion

We have built a complete system based on the GRASP approach from [1]. We have combined the core SPARQL question-answering capability with general question answering and multi-turn follow-up questions. We have extended the evaluation from [1] by new experiments on the IMDb knowledge graph and the TEXT2SPARQL challenge, with strong results. This provides further support for GRASP's zero-shot capabilities across knowledge graphs.

For future work, we consider the support of automatic builds of search indices from nothing but a SPARQL endpoint, as well as integrating search indices and their functionality directly into SPARQL. This would be a step towards both *zero-shot* and *zero-configuration* question answering on arbitrary given knowledge graphs. To prevent GRASP from repeatedly making the same mistakes, which can occur in the zero-shot setting, we also consider adding memory or other forms of online learning as potential future work.

Acknowledgments

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

Declaration on Generative AI

The author(s) have not employed any Generative AI tools.

References

[1] S. Walter, H. Bast, GRASP: Generic reasoning and SPARQL generation across knowledge graphs, in: ISWC, 2025. Accepted for publication. Preprint available at https://arxiv.org/abs/2507.08107.

[2] D. Vrandecic, M. Krötzsch, Wikidata: a free collaborative knowledgebase, Commun. ACM 57 (2014) 78–85.

[3] K. D. Bollacker, C. Evans, P. K. Paritosh, T. Sturge, J. Taylor, Freebase: a collaboratively created graph database for structuring human knowledge, in: SIGMOD Conference, ACM, 2008, pp. 1247–1250.

[4] M. R. Ackermann, H. Bast, B. M. Beckermann, J. Kalmbach, P. Neises, S. Ollinger, The dblp knowledge graph and SPARQL endpoint, TGDK 2 (2024) 3:1–3:23. URL: https://doi.org/10.4230/TGDK.2.2.3. doi:10.4230/TGDK.2.2.3.

[5] IMDb, IMDb: Ratings, Reviews, and Where to Watch the Best Movies & TV Shows, 2025. URL: https://www.imdb.com/.

[6] AKSW, TEXT2SPARQL'25 - AKSW, 2025. URL: https://text2sparql.aksw.org/.

[7] D. Yu, S. Zhang, P. Ng, H. Zhu, A. H. Li, J. Wang, Y. Hu, W. Y. Wang, Z. Wang, B. Xiang, DecAF: Joint decoding of answers and logical forms for question answering over knowledge bases, in: ICLR, OpenReview.net, 2023.

[8] L. Luo, Y. Li, G. Haffari, S. Pan, Reasoning on graphs: Faithful and interpretable large language model reasoning, in: ICLR, OpenReview.net, 2024.

[9] H. Luo, H. E, Z. Tang, S. Peng, Y. Guo, W. Zhang, C. Ma, G. Dong, M. Song, W. Lin, Y. Zhu, A. T. Luu, ChatKBQA: A generate-then-retrieve framework for knowledge base question answering with fine-tuned large language models, in: ACL (Findings), Association for Computational Linguistics, 2024, pp. 2039–2056.

[10] M. Patidar, R. Sawhney, A. K. Singh, B. Chatterjee, Mausam, I. Bhattacharya, Few-shot transfer learning for knowledge base question answering: Fusing supervised models with in-context learning, in: ACL (1), Association for Computational Linguistics, 2024, pp. 9147–9165.

[11] Y. Gu, X. Deng, Y. Su, Don't generate, discriminate: A proposal for grounding language models to real-world environments, in: ACL (1), Association for Computational Linguistics, 2023, pp. 4928–4949.

[12] J. Ma, Z. Gao, Q. Chai, W. Sun, P. Wang, H. Pei, J. Tao, L. Song, J. Liu, C. Zhang, L. Cui, Debate on graph: A flexible and reliable reasoning framework for large language models, in: AAAI, AAAI Press, 2025, pp. 24768–24776.

[13] S. Liu, S. J. Semnani, H. Triedman, J. Xu, I. D. Zhao, M. S. Lam, SPINACH: SPARQL-based information navigation for challenging real-world questions, in: EMNLP (Findings), Association for Computational Linguistics, 2024, pp. 15977–16001.

[14] J. Sun, C. Xu, L. Tang, S. Wang, C. Lin, Y. Gong, L. M. Ni, H. Shum, J. Guo, Think-on-Graph: Deep and responsible reasoning of large language model on knowledge graph, in: ICLR, OpenReview.net, 2024.

[15] J. Jiang, K. Zhou, Z. Dong, K. Ye, X. Zhao, J. Wen, StructGPT: A general framework for large language model to reason over structured data, in: EMNLP, Association for Computational Linguistics, 2023, pp. 9237–9251.

[16] OpenAI, Introducing GPT-4.1 in the API, 2025. URL: https://openai.com/index/gpt-4-1/, accessed: 2025-05-11.

[17] Google DeepMind, Introducing Gemini 2.0: our new AI model for the agentic era, 2024. URL: https://blog.google/technology/google-deepmind/google-gemini-ai-update-december-2024/, accessed: 2025-05-11.

[18] 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).

[19] Qwen Team, Qwen3 technical report, 2025. URL: https://arxiv.org/abs/2505.09388. arXiv:2505.09388.

[20] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. E. Gonzalez, H. Zhang, I. Stoica, Efficient

memory management for large language model serving with pagedattention, in: Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles, 2023.

[21] A. Talmor, J. Berant, The web as a knowledge-base for answering complex questions, in: NAACL-HLT, Association for Computational Linguistics, 2018, pp. 641–651.

[22] W. Yih, M. Richardson, C. Meek, M. Chang, J. Suh, The value of semantic parse labeling for knowledge base question answering, in: ACL (2), The Association for Computer Linguistics, 2016.

[23] R. Usbeck, X. Yan, A. Perevalov, L. Jiang, J. Schulz, A. Kraft, C. Möller, J. Huang, J. Reineke, A.-C. 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://journals.sagepub.com/doi/abs/10.3233/SW-233471. doi:10.3233/SW-233471. arXiv:https://journals.sagepub.com/doi/pdf/10.3233/SW-233471.

[24] 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: SemWebEval@ESWC, volume 769 of *Communications in Computer and Information Science*, Springer, 2017, pp. 59–69.

[25] S. Xu, S. Liu, T. Culhane, E. Pertseva, M. Wu, S. J. Semnani, M. S. Lam, Fine-tuned LLMs know more, hallucinate less with few-shot sequence-to-sequence semantic parsing over Wikidata, in: EMNLP, Association for Computational Linguistics, 2023, pp. 5778–5791.

[26] Y. Zhang, M. Li, D. Long, X. Zhang, H. Lin, B. Yang, P. Xie, A. Yang, D. Liu, J. Lin, F. Huang, J. Zhou, Qwen3 embedding: Advancing text embedding and reranking through foundation models, arXiv preprint arXiv:2506.05176 (2025).

[27] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, Z. G. Ives, DBpedia: A nucleus for a web of open data, in: ISWC/ASWC, volume 4825 of *Lecture Notes in Computer Science*, Springer, 2007, pp. 722–735.

[28] M. Y. Jaradeh, A. Oelen, K. E. Farfar, M. Prinz, J. D'Souza, G. Kismihók, M. Stocker, S. Auer, Open research knowledge graph: Next generation infrastructure for semantic scholarly knowledge, in: K-CAP, ACM, 2019, pp. 243–246.

[29] H. Bast, P. Brosi, J. Kalmbach, A. Lehmann, An efficient RDF converter and SPARQL endpoint for the complete OpenStreetMap data, in: SIGSPATIAL/GIS, ACM, 2021, pp. 536–539.

[30] L. Garcia, J. Bolleman, S. Gehant, N. Redaschi, M. Martin, A. Bateman, M. Magrane, S. Orchard, S. Raj, S. Ahmad, E. Alpi, E. Bowler, R. Britto, B. Bursteinas, H. Bye-A-Jee, T. Dogan, P. Garmiri, G. Georghiou, L. Gonzales, E. Hatton-Ellis, A. Ignatchenko, G. Insana, R. Ishtiaq, V. Joshi, D. Jyothi, J. Luo, Y. Lussi, A. MacDougall, M. Mahmoudy, A. Nightingale, C. Oliveira, J. Onwubiko, V. Poddar, S. Pundir, G. Qi, A. Rifaioglu, D. Rice, R. Saidi, E. Speretta, E. Turner, N. Tyagi, P. Vasudev, V. Volynkin, K. Warner, X. Watkins, R. Zaru, H. Zellner, A. Bridge, L. Breuza, E. Coudert, D. Lieberherr, I. Pedruzzi, S. Poux, M. Pruess, L. Aimo, G. Argoud-Puy, A. Auchincloss, K. Axelsen, P. Bansal, D. Baratin, T. Batista Neto, M.-C. Blatter, E. Boutet, C. Casals-Casas, B. Cuche, E. De Castro, A. Estreicher, L. Famiglietti, M. Feuermann, E. Gasteiger, V. Gerritsen, A. Gos, N. Gruaz, U. Hinz, C. Hulo, N. Hyka-Nouspikel, F. Jungo, A. Kerhornou, P. Lemercier, T. Lombardot, P. Masson, A. Morgat, S. Pilbout, M. Pozzato, C. Rivoire, C. Sigrist, S. Sundaram, C. Wu, C. Arighi, H. Huang, P. McGarvey, D. Natale, L. Arminski, C. Chen, Y. Chen, J. Garavelli, K. Laiho, K. Ross, C. R. Vinayaka, Q. Wang, Y. Wang, L.-S. Yeh, J. Zhang, U. Consortium, FAIR adoption, assessment and challenges at UniProt, Scientific Data 6 (2019) 175. URL: https://doi.org/10.1038/s41597-019-0180-9. doi:10.1038/s41597-019-0180-9.

[31] D. Banerjee, S. Awale, R. Usbeck, C. Biemann, DBLP-QuAD: A question answering dataset over the DBLP scholarly knowledge graph, in: BIR@ECIR, volume 3617 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2023, pp. 37–51.