Instruct-to-SPARQL: A text-to-SPARQL dataset for training Wikidata Agents

Alexis Strappazon, Michael Granitzer, Előd Egyed-Zsigmond, Jelena Mitrovic, Mehdi Ben Amor

Instruct-to-SPARQL: A text-to-SPARQL dataset for training Wikidata Agents

Mehdi Ben Amor*
mehdi.benamor@uni-passau.de
University of Passau
Passau, Germany

Alexis Strappazon*
alexis.strappazzon@insa-lyon.fr
INSA Lyon
Lyon, France

Michael Granitzer
University of Passau
Passau, Germany
michael.granitzer@uni-passau.de

Előd Egyed-Zsigmond
LIRIS, INSA Lyon
Lyon, France

Jelena Mitrović
University of Passau
Passau, Germany

Abstract

The rapid adoption of Large Language Models (LLMs) for search engines and fact-checking platforms necessitates enhancing their output accuracy. Retrieval Augmented Generation (RAG) mitigates hallucinations but requires semantically rich repositories like Wikidata. However, there is a lack of high-quality data to fine-tune LLMs for querying such knowledge bases. To address this gap, we propose a curated dataset with 2,771 unique queries for fine-tuning LLMs to generate accurate and syntactically valid SPARQL queries from natural language instructions. This dataset, customized for interaction with Wikidata, also serves as a robust benchmark for text-to-SPARQL task evaluation. Key findings show that models generally perform better on queries with lower complexity.

CCS Concepts

• **Information systems** → **Information retrieval**; **Question answering**.

Keywords

SPARQL, Information Retrieval, Large Language Models, Text-to-SPARQL

ACM Reference Format:

Mehdi Ben Amor, Alexis Strappazon, Michael Granitzer, Előd Egyed-Zsigmond, and Jelena Mitrović. 2025. Instruct-to-SPARQL: A text-to-SPARQL dataset for training Wikidata Agents. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (CHIIR '25).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

Text-to-**Query** is a task that aims to automatically generate structured queries that communicate with various data and knowledge base stores. It bridges the natural language (NL) domain of unstructured data with rigorously structured data sources. The task of parsing natural language questions into structured queries is even more prevalent now, given the high potential that Large Language Models (LLMs) offer in Retrieval-Augmented conversational agents and the problems they face with using the correct information as context.

Given the accessibility and diversity of Structured Query Language or shortly *SQL* in relational databases, text-to-SQL is currently one of the most challenging and essential tasks in semantic parsing. Spider [11] is an example of a large-scale dataset used for cross-domain semantic parsing and text-to-SQL tasks. There are many other datasets for this task, such as GenQuery [13], WikiSQL [14], and CoSQL [12], to name a few. Text-to-SQL datasets have been categorized into two types: single-turn and multi-turn. As essential as SQL is, it is only useful for querying relational databases. Relational data consists of rows collected into tables that have to conform to a fixed set of constraints and data types, i.e., a "schema".

SPARQL[1] for SPARQL Protocol and RDF Query Language is designed to generate queries on data that does not follow a fixed schema. It is a semantic query language able to retrieve and manipulate data stored in RDF format or triplets of subject, predicate, and object. It offers more flexibility and semantic richness, with the key advantage of having multiple sources of information and endpoints to query from. It is also worth noting that SPARQL adds a level of complexity compared to SQL in the task of query generation since the former needs proper ID mapping or else they fail easily. Kosten et al. introduced a text-to-SPARQL benchmark called SPIDER4SPARQL [7] where they converted the Spider dataset and databases to NL/SPARQL pairs and knowledge graphs. Their benchmark translates the variety of domains and queries from Spider. However, it relies on synthetically created RDF stores that are a direct mapping from the relational DBs. Other works like *DBNQA* [8], *LC-QuAD 2.0* [3] are also extensive benchmarks for Knowledge Graphs Question Answering or text-to-SPARQL that were generated using a set of SPARQL query templates. While these benchmarks provide valuable resources, they are limited by their reliance on synthetic query generation and template-based approaches. This results in a significant gap in the field - the lack of datasets containing real-world SPARQL queries that capture the full complexity and variability found in practice. Real SPARQL queries,

*Both authors contributed equally to this research.

CHIIR '25, June 03–05, 2025, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/18/06
https://doi.org/10.1145/nnnnnnn.nnnnnnn

[1] https://www.w3.org/TR/sparql11-query/

drawn from actual usage, tend to exhibit more sophisticated patterns and edge cases that are difficult to replicate through synthetic generation methods.

To address this gap, we introduce *Instruct-to-SPARQL*[2], a text-to-SPARQL dataset aimed at Wikidata that consists of queries collected from real examples and tutorials on querying Wikidata. Moreover, we tackle the issue of entity ID mapping, which is caused by a major flaw of text generation models (LLMs) - the "hallucination" phenomenon where generated queries may contain non-existent or outdated IDs upon generation. The dataset can be used as a benchmark to evaluate the capabilities of LLMs to query Wikidata or to fine-tune models that can be used as retrieval components for Wiki Chatbots.

Our main contributions in this work are as follows:

(1) An Instruct-to-SPARQL dataset for Wikidata with 13,855 generated NL instructions and 2,771 unique SPARQL queries with various complexity levels and entity ID annotations.
(2) *SPARQL-LMs*: We release the weights of the fine-tuned LLMs for the SPARQL query generation task.

The paper is organized as follows: Section 2 presents the methodology used to collect and curate the dataset. Section 3 describes the Instruct-to-SPARQL dataset with the additional instructions generation, Query labeling and complexity classification. Section 4 presents the potential applications of the dataset and a small experimental setup and evaluation. Finally, Section 5 concludes the paper and discusses future work.

2 Methodology

2.1 Data Collection

The dataset was collected by crawling Wikidata weekly examples pages[3], tutorials[4], and other web pages[5] containing examples of SPARQL queries for Wikidata. We focused on crawling the queries and their surrounding text, including headings and descriptions as relevant "context". During crawling, we did not distinguish between new and previously seen queries from different pages. After crawling, we obtained 6,739 queries along with their contextual metadata.

2.2 Data Cleaning and Validation

Given the challenging task of query generation, language models require high-quality training data. Therefore, we sanitized the crawled data and removed duplicates. Our data cleaning process involved:

- Deduplication: We reduced 6,739 initial entries to **2,771 unique queries** through a two-step process. First, we identified and removed exact duplicates that had identical crawled queries and attached context. Then, we manually reviewed the remaining query duplicates, retaining those with the most informative context.
- Context cleaning: The retrieved context often contained HTML markup tags, CSS styling code, and long URLs, which we deemed irrelevant and removed. By cleaning and removing noisy text from the context, we aimed to improve the quality of our natural language instruction generations.
- Query validation: We executed all queries through Wikidata's SPARQL endpoint to ensure syntactic correctness.

Error Type	Description
Client-side Errors	
Undefined prefix	Missing prefix declaration in query
Parse exception	Invalid SPARQL syntax
Lexical exception	Presence of illegal characters
Non-aggregate error	Invalid variable usage in SELECT clause
Server-side Errors	
Stack overflow	Excessive query recursion depth
GeoSpatial error	Failed coordinate resolution
Timeout	Query execution time exceeded limit
Out of memory	Memory allocation exceeded
NullPointer	Access to non-existent resource

Table 1: Taxonomy of errors encountered during SPARQL query validation on Wikidata's endpoint.

2.3 Validation and Error Analysis

While the context provides the natural language explanation of a query, the query itself must be syntactically correct (i.e., executable). Due to the inherent complexity of automated semantic validation, we utilized the presence of a non-empty result set as a proxy indicator for semantic validity, though this heuristic provides only partial confidence in correctness.

We used Wikidata's SPARQL Endpoint API[6] to execute all 2,771 unique queries. We classified errors into client-side (syntax, parsing) and server-side (timeout, memory) categories, as shown in Table 1.

From the initial dataset:

- 2,415 queries executed successfully
- 109 returned empty results
- 178 encountered server-side errors
- 77 encountered client-side errors

Among the 77 client-side errors, we observed the following distribution: 33 lexical errors (42.9%), 27 undefined prefix errors (35.1%), 16 parse exceptions (20.8%), and 1 non-aggregate error (1.3%).

We addressed various client-side errors through targeted fixes. For *undefined prefix* errors (27 cases), which occurred due to missing declarations for external RDF stores and non-standard prefix abbreviations, we inserted the required prefixes and standardized the abbreviations. *Lexical errors* were resolved by properly escaping special characters in string filters, while *parse exceptions* stemming from incomplete queries due to scraping errors were manually corrected where possible by referring to the original source. Server-side errors, such as timeouts and memory limitations, remained unmodified as query optimization was beyond this work's scope.

[2]https://github.com/padas-lab-de/instruct-to-sparql
[3]https://www.wikidata.org/wiki/Wikidata:Weekly_query_examples
[4]https://www.wikidata.org/wiki/Wikidata:SPARQL_tutorial
[5]https://www.wikidata.org/wiki/User:MartinPoulter/queries/botany
[6]https://query.wikidata.org/

Algorithm 1 Labeling algorithm (for one query)

Input: q a SPARQL query
  $t \leftarrow q$
  $I \leftarrow$ Extract a set of all Wikidata's Identifier from q
  for each $i \in \mathcal{I}$ **do**
    $L \leftarrow$ Retrieve labels of i using Wikidata's API
    if L has an English label **then**
      $l \leftarrow$ The English labels from L
    else if L is not empty **then**
      $l \leftarrow$ The first label from L
    else
      Next iteration
    end if
    $t \leftarrow$ Replace all occurrences of i with the type of item and l
  end for
Output: t is a labeled q

Query	SELECT (COUNT(DISTINCT ?article) AS ?count) WHERE {?article wdt:P31/wdt:P279* wd:Q95074}
Query Templated	SELECT (COUNT(DISTINCT ?article) AS ?count) WHERE {?article wdt:[property:instance of]/wdt:[property:subclass of]* wd:[entity:fictional character]}

Table 2: An example of a query and the same query with IDs annotated.

Split	**train (1.85k)**	**validation (124)**	**test (495)**
`Simple`	6.2%	4.9%	6.2%
`Moderate`	40.9%	47.2%	40.8%
`Complex`	52.9%	48%	53%

Table 3: SPARQL query complexity distribution across data splits.

3 Instruct-to-SPARQL Dataset

Our dataset consists of 2,771 unique SPARQL queries paired with 13,855 natural language instructions (see Table 4). Key features include:

- Query complexity labels (simple, moderate, complex)
- Entity and property annotations
- Natural language instructions for each query

3.1 Query Labeling Process

To enhance the semantic interpretability and robustness of our dataset, we introduce a systematic approach for replacing Wikidata's opaque identifiers with meaningful English labels. This transformation addresses several key challenges in SPARQL query generation:

First, Wikidata's identifier system (using "Q" for entities, "P" for properties, and "L" for lexemes followed by numbers) is inherently non-intuitive for both language models and humans. These identifiers lack semantic meaning and can change over time as Wikidata evolves. By replacing them with descriptive English labels, we create a more stable and semantically rich representation.

Second, this labeling approach better aligns with how humans naturally express queries, bridging the gap between natural language and structured query languages. Rather than requiring models to learn arbitrary identifier mappings, they can work with meaningful labels that directly convey semantic intent.

Our labeling process, detailed in Algorithm 1, systematically transforms each query while preserving its semantic structure. To handle the non-unique nature of labels (unlike identifiers), we implemented a type-prefixing system. For instance, when both a property (P31) and entity (Q21503252) share the label "Instance Of", we disambiguate them by prefixing with their types (e.g., "property:Instance Of" vs. "entity:Instance Of"). This ensures unambiguous interpretation while maintaining semantic clarity.

We acknowledge certain limitations in our approach. When processing identifiers that do not exist in Wikidata (either due to deletion or modification), we leave them unchanged rather than attempting potentially incorrect mappings. While this means some identifiers remain unlabeled, it preserves query integrity. Additionally, in cases where an identifier is misinterpreted and mapped to an incorrect Wikidata item, the query's semantics could be altered. However, we prioritize maintaining a high-quality dataset over attempting uncertain historical reconstructions of modified or deleted items.

This labeled representation transforms the query generation task from exact identifier prediction to label generation, which we hypothesize is more amenable to LLM capabilities. While this introduces the need for a subsequent label-to-identifier linking step, it creates a more natural intermediate representation that better leverages the semantic understanding capabilities of large language models.

This process of labeling is automated using the Wikidata API and can be used on any SPARQL query for Wikidata. An example of labeled query is given in Table 2. Labeling changes the problem from generating the exact identifier to generating labels. Nevertheless, it implies a new system that would link labels back to item identifiers is **needed**, as the query cannot be executed with labels. The Wikidata API has the functionality to search items by labels to get a list of candidates. We do not propose any linking strategies in this work, we invite future work to address this challenge.

3.2 Natural Language Instructions

After annotating the queries, we used them to generate natural language user instructions or questions. The enriched semantic representation of the annotated SPARQL queries, combined with the crawled "context", led to higher quality instruction generation. We generated five natural language instructions or questions for each query using **Llama3-70B** [1].

3.3 Query Complexity Classification

Building upon our labeled query representation, we further enhanced the dataset by classifying queries according to their complexity. Using Llama3-70B, we assessed each query and assigned it one of three complexity levels: *simple*, *moderate*, or *complex*. This classification provides valuable metadata for both training and

Field	Content
Instructions	"Find all movies directed by Christopher Nolan" "..."

Query	`SELECT ?movie WHERE {`
	`  ?movie wdt:P31 wd:Q11424.`
	`  ?movie wdt:P57 wd:Q25191.`
	`}`

Annotated	`SELECT ?movie WHERE {`
	`  ?movie wdt:[property:instance of]`
	`  wd:[entity:film].`
	`  ?movie wdt:[property:director]`
	`  wd:[entity:Christopher Nolan].`
	`}`

Complexity	simple
Results	`[{movie: Interstellar},` `{movie: The Dark Knight}, ...]`

Table 4: Example entry from the Instruct-to-SPARQL dataset.

evaluation purposes. When partitioning the dataset into train, validation, and test splits, we applied two key criteria: (1) we included only queries that were successfully executed and returned results, ensuring practical utility, and (2) we maintained a balanced distribution of complexity levels across all splits, as shown in Table 3. This careful stratification ensures that model performance can be meaningfully evaluated across different complexity levels.

3.4 Dataset Applications

4 Benchmark Evaluation

We evaluated our dataset on two primary tasks that assess different aspects of SPARQL query generation capabilities:

Text-to-SPARQL Generation. This base task evaluates the model's ability to generate syntactically correct SPARQL queries from natural language instructions. This represents the fundamental challenge of translating user intent into executable queries. Models must not only understand the semantic meaning of the instruction but also produce valid SPARQL syntax with correct Wikidata identifiers.

Labeled Query Generation. This task evaluates our proposed approach of using natural language labels instead of opaque Wikidata identifiers. Models generate queries using human-readable labels (e.g., "entity:United States" instead of "Q30"), which are then mapped back to Wikidata IDs using a *label-to-identifier* matching approach. This tests whether semantic representations improve generation quality while maintaining query executability through post-processing.

4.1 Baselines

We evaluated our dataset using two medium-sized large language models (Mistral and Llama3) via fine-tuning, and Llama3-70B, GPT-3.5, and GPT-4 via few-shot prompting. For fine-tuning, we trained the models for 3 epochs with a batch size of 96 for a total of 174 steps using the full train and validation sets of the Instruct-to-SPARQL dataset.[7] We released the source code for the dataset collection and fine-tuning experiments on GitHub[8].

- **Mistral-7B-v0.3** [6]: A capable open-source language model suitable for many text generation tasks. We used the instruct fine-tuned version.
- **Llama3-8B** [1, 9]: The latest open-source release of the Llama series from Meta. We used the instruction-tuned version.
- **Llama3-70B-Instruct** [1, 9]: We selected this model for few-shot learning as it is the most capable open-source model. We randomly selected two examples from the training set for few-shot prompting.
- **GPT-3.5** [2]: We used the OpenAI API to generate queries for the text-to-SPARQL task.
- **GPT-4** [4]: We used the OpenAI API to generate queries for the text-to-SPARQL task.

4.2 Evaluation Metrics

We evaluated the models using two sets of metrics: **Text Generation** We used **CodeBLEU**[9] [10] to evaluate the similarity between the generated SPARQL queries and the target queries. **Query Execution** We evaluated the **Jaccard similarity** between the sets of target results and generated query results[10]. The **Syntax score** represents the ratio of syntactically valid and correct queries to all generated queries.

4.3 Results and Analysis

For this experimental setup, we compared training directly on queries with IDs against training with annotated queries where we obfuscated the entity and property IDs with labels.

Table 5 reveals several key findings. First, supervised fine-tuning (SFT) significantly outperforms few-shot learning across all metrics for both model types. In the non-annotated setting, SFT models achieved CodeBLEU scores of 84-85% and Jaccard similarities of 70%, compared to few-shot scores below 56% and 30% respectively. Second, contrary to our initial hypothesis, models fine-tuned on non-annotated queries (with direct IDs) achieved better CodeBLEU scores (84-85%) compared to those trained on annotated queries (62-64%). However, the Syntax scores remained consistently high (96%) across both settings with SFT. Third, while GPT-4 showed the strongest few-shot performance (particularly in Syntax score at 95.3%), it was still substantially outperformed by fine-tuned models in terms of result accuracy (Jaccard similarity). Notably, larger models like Llama3-70B performed poorly in few-shot settings,

[7] All fine-tuning experiments were conducted on 6x A100 GPUs (80GB memory) with an AMD EPYC 7742 64-Core Processor and 200GB memory.
[8] https://github.com/padas-lab-de/instruct-to-sparql
[9] We used a modified version where we compute the metric on the parsed SPARQL query.
[10] For this, we applied a semantic mapping between the schema keys in the results JSON objects [5]

Table 5: Evaluation results on the test set of Instruct-to-SPARQL on the two tasks.

Text-to-SPARQL			
	CodeBLEU	**Syntax score**	**Jaccard**
Few-shots			
Llama3-8B	48.5%	52.1%	2.0%
Mistral-7B	39.9%	33.8%	1.1%
Llama3-70B	45.3%	14.1%	1.6%
GPT3.5	48.2%	94.4%	18.4%
GPT4	56.0%	95.3%	29.9%
SFT			
Llama3-8B	84.4%	96.0%	70.4%
Mistral-7B	85.0%	96.7%	70.8%
Labeled Query Generation			
	CodeBLEU	**Syntax score**	**Jaccard**
Few-shots			
Llama3-8B	36.4%	67.9%	7.2%
Mistral-7B	40.3%	61.1%	6.8%
Llama3-70B	45.3%	37.8%	10.8%
GPT3.5	24.4%	96.2%	21.5%
GPT4	42.9%	90.5%	30.4%
SFT			
Llama3-8B	62.5%	96.0%	69.0%
Mistral-7B	63.9%	96.5%	72.5%

achieving only 14.1% and 37.8% Syntax scores for non-annotated and annotated queries respectively, suggesting that model size alone is insufficient for complex SPARQL query generation without task-specific training.

4.4 Complexity-based Performance

We analyzed the performance of these models across different complexity levels of SPARQL queries in our dataset. Figure 1 illustrates the effect of query complexities on results-based performance using the Jaccard score.

5 Conclusion

In this paper, we introduced Instruct-to-SPARQL, a curated dataset of SPARQL queries paired with natural language instructions designed to serve as a benchmark for text-to-SPARQL tasks on Wikidata. To address the issue of generating unique IDs for Wikidata entities, we implemented an annotation method utilizing natural language labels, thereby enhancing the reliability of LLMs in generating accurate queries. Moving forward, we plan to refine the process of ID label replacements post-query generation to further enhance our approach.

References

[1] AI@Meta. 2024. Llama 3 model card. (2024). https://github.com/meta-llama/llama3/blob/main/MODEL_CARD.md

[2] Tom B. et al. Brown. 2020. Language Models are Few-Shot Learners. https://doi.org/10.48550/arXiv.2005.14165 arXiv:2005.14165.

[3] Mohnish Dubey, Debayan Banerjee, Abdelrahman Abdelkawi, and Jens Lehmann. 2019. Lc-quad 2.0: A large dataset for complex question answering over wikidata and dbpedia. In *The semantic web–ISWC 2019: 18th international semantic web conference, auckland, new zealand, october 26–30, 2019, proceedings, part II 18*. Springer, 69–78.

[4] OpenAI et al. 2024. GPT-4 Technical Report. https://doi.org/10.48550/arXiv.2303.08774 arXiv:2303.08774.

[5] fireinfark707. 2022. Schema matching by xgboost. https://github.com/fireindark707/Schema_Matching_XGboost

[6] AQ Jiang, A Sablayrolles, A Mensch, C Bamford, DS Chaplot, D de las Casas, F Bressand, G Lengyel, G Lample, L Saulnier, and others. 2023. Mistral 7B (2023). *arXiv preprint arXiv:2310.06825* (2023).

[7] Catherine Kosten, Philippe Cudré-Mauroux, and Kurt Stockinger. 2023. Spider4SPARQL: a complex benchmark for evaluating knowledge graph question answering systems. In *2023 IEEE international conference on big data (BigData)*. IEEE, 5272–5281.

[8] Trond Linjordet and Krisztian Balog. 2020. Sanitizing synthetic training data generation for question answering over knowledge graphs. In *Proceedings of the 2020 ACM SIGIR on international conference on theory of information retrieval*.

[9] AI Meta. 2024. Introducing meta llama 3: The most capable openly available llm to date, 2024. *URL https://ai. meta. com/blog/meta-llama-3/*. 26 (2024).

[10] Shuo Ren, Daya Guo, Shuai Lu, Long Zhou, Shujie Liu, Duyu Tang, Neel Sundaresan, Ming Zhou, Ambrosio Blanco, and Shuai Ma. 2020. CodeBLEU: a Method for Automatic Evaluation of Code Synthesis. https://doi.org/10.48550/arXiv.2009.10297 arXiv:2009.10297.

[11] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 2018. Spider: a large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii (Eds.). Association for Computational Linguistics, Brussels, Belgium, 3911–3921. https://doi.org/10.18653/v1/D18-1425

[12] Tao et al. Yu. 2019. CoSQL: a conversational text-to-SQL challenge towards cross-domain natural language interfaces to databases. In *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)*, Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan (Eds.). Association for Computational Linguistics, Hong Kong, China, 1962–1979. https://doi.org/10.18653/v1/D19-1204

[13] John M Zelle and Raymond J Mooney. 1996. Learning to parse database queries using inductive logic programming. In *Proceedings of the national conference on artificial intelligence*. 1050–1055.

[14] Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2sql: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103* (2017).

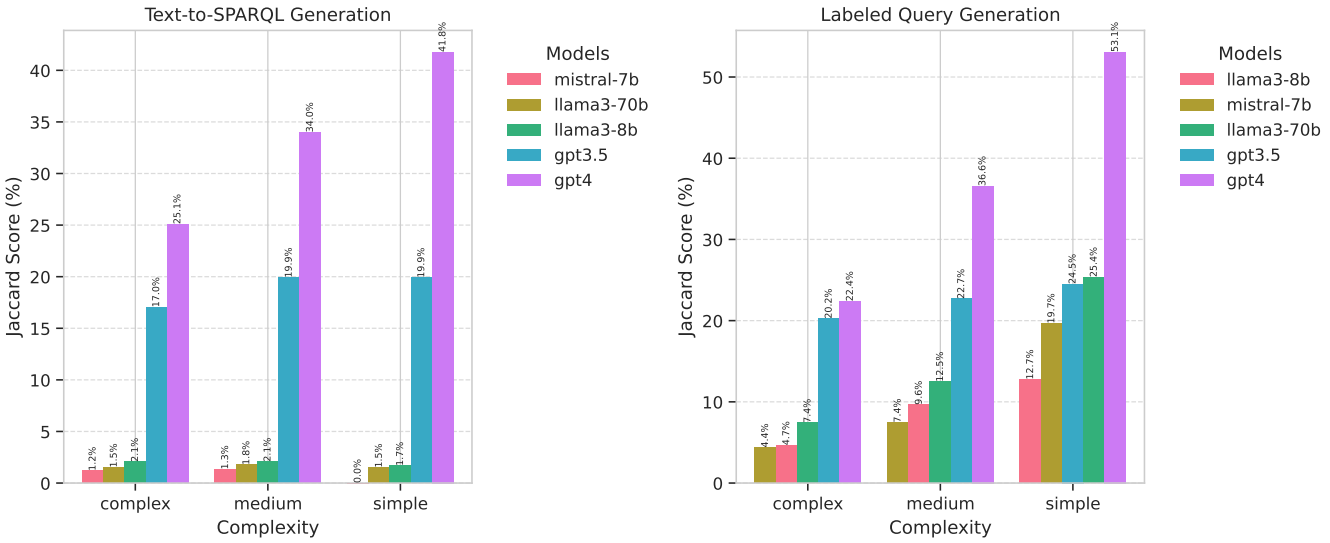

Figure 1: Average Jaccard score of all models based on complexity.