Data Engineering Research Group
TUM School of Computation, Information and Technology
Technical University of Munich

Querying Wikidata with Natural Language

Bachelor Practical Course on Data Engineering (SoSe 2025)

Xaver Baumgartner ✉ **and Yun Zhou** ✉

TUM School of Computation, Information and Technology, Technical University of Munich

✉ Xaver.Baumgartner@tum.de
✉ go72fig@mytum.de

July 9, 2025

Abstract — This report details the development of 'Wikiquery', a web app designed for natural language querying of the Wikidata knowledge base. The system translates user prompts into SPARQL (SPARQL Protocol and RDF Query Language) queries using Large Language Models (LLMs), specifically variants of Google's Gemini 2.5 Flash. Four distinct approaches were explored: a direct Gemini implementation, a version that was finetuned using SPARQL queries with natural language placeholders, and two Model Context Protocol (MCP) enhanced variants (one based on the standard and one based on the finetuned model). These models were evaluated using a benchmark dataset of queries with known solutions to assess their performance, strengths, and weaknesses. The MCP approach, which allows the LLM to interact with Wikidata's search and query APIs during the generation process, providing dynamic context and enabling self-correction, proved to be the most performant in addressing the crucial challenge of making complex knowledge bases accessible to non-technical users, thereby democratizing data access.

1 Introduction

1.1 Problem Motivation

In today's data-driven world, knowledge bases like Wikidata serve as invaluable repositories of structured information. However, accessing this information typically requires proficiency in query languages such as SPARQL (SPARQL Protocol and RDF Query Language), which limits accessibility for the general public and domain experts without programming expertise. This project aims to democratize data access by enabling users to retrieve this information using natural language.

1.2 Current Challenges

The primary challenge in this domain is bridging the gap between ambiguous and diverse human language and the strict syntax and precise entity/property identification required by formal query languages. This includes:

- **Semantic Gap:** Translating the nuanced meaning of natural language into the exact semantics required by SPARQL queries is difficult, especially for sloppily written prompts. An example is distinguishing "country" as a nation-state from "country" as a music genre.
- **Query Complexity:** Generating complex SPARQL queries involving multiple hops, temporal constraints, aggregations, or unions is challenging for LLMs without explicit guidance or iterative refinement.
- **Entity matching:** LLMs often lack real-time access to external knowledge bases. This limitation can lead to "hallucinations" where the LLM invents database IDs that do not correspond to actual, relevant entities.
- **Evaluation Metrics:** Quantifying the "correctness" of a generated SPARQL query is non-trivial, as a syntactically correct query might yield semantically incorrect results, and query equivalence cannot be determined algorithmically.

2 Preliminaries

2.1 Wikidata

Wikidata is a free and open knowledge base that acts as a central storage for the structured data of its Wikimedia sister projects, including Wikipedia, Wikivoyage, Wiktionary, etc. Wikidata uses a triple-store model, where data is stored as subject-predicate-object triples. Items (subjects and objects) are identified by Q-numbers (e.g., Q1339 for Johann Sebastian Bach), and properties are identified by P-numbers (e.g., P40 for child).

2.2 SPARQL

SPARQL (SPARQL Protocol and RDF Query Language) is a query language for graph databases using RDF (Resource Description Framework), including Wikidata. A basic SPARQL query consists of a SELECT clause, specifying what to retrieve, and a WHERE clause, specifying the graph patterns to match. For example, to find the children of Johann Sebastian Bach, one might use `'SELECT ?child WHERE {wd:Q1339 wdt:P40 ?child . }'`.

2.3 Large Language Models (LLMs)

LLMs are advanced deep learning models trained on vast amounts of text data, capable of tasks such as text generation, summarization, and question answering. For this project, Google's Gemini 2.5 Flash model was chosen for being highly efficient while maintaining strong performance in complex reasoning tasks.

2.4 Model Context Protocol (MCP)

The Model Context Protocol (MCP) enables LLMs to interact with external tools and APIs during the generation process. Instead of generating a complete response all at once, an MCP-enabled LLM can make function calls (e.g., to a search API or a database query tool), receive the results, and refine its subsequent outputs based on this information. This iterative process is crucial for tasks requiring real-time data retrieval, validation, and self-correction, as it provides dynamic context that cannot be inferred from initial training data.

3 Problem Statement

The core problem addressed is how to effectively and accurately translate natural language questions into executable SPARQL queries for the Wikidata knowledge base, enabling non-technical users to access structured data without learning SPARQL. The hypothesis is that leveraging advanced LLMs, especially those with contextual interaction capabilities, can significantly improve the accuracy and robustness of natural language to SPARQL query translation compared to simpler LLM integrations. The questions to be researched are:

1. How does a direct LLM-based translation approach (with post-processing for ID resolution) perform in generating SPARQL queries for Wikidata?

2. Can finetuning the LLM on a dataset of natural language inputs and SPARQL query (with ID placeholders) outputs improve translation accuracy?

3. What is the impact of integrating Model Context Protocol (MCP) tools, allowing the LLM to interact with Wikidata APIs during query generation, on the accuracy of the generated SPARQL queries?

4 Main Approach

We implemented and evaluated four distinct approaches, all based on Gemini 2.5 Flash. The complete source code is available on GitHub[2].

4.1 Gemini 2.5 Flash (Direct)

This is the simplest approach.

- **LLM Role:** The Gemini 2.5 Flash model is prompted to directly generate a SPARQL-like query. Crucially, instead of using actual Wikidata IDs, it generates symbolic placeholders (e.g., wd:France instead of wd:Q142).

- **Post-processing Script:** The output is parsed using the Wikidata Search API to find the most relevant Wikidata ID for each placeholder. For example, wd:France would be replaced with wd:Q142.

- **Contribution to Problem Solution:** There are over 110 million Wikidata IDs[1], many of which are not in the training data. An even more direct implementation, without the placeholder post-processing concept, inevitably delivers hallucinated IDs and therefore generally no useful results.

4.2 Gemini (Finetuned)

This approach builds upon the direct Gemini 2.5 Flash model by incorporating finetuning.

- **Finetuning Dataset:** The model is finetuned using a column subset of the `lc_quad_synth` [4] dataset, which is an enhanced version of `Lc-quad 2.0` [3]. It contains input-output tuples, where inputs are natural language questions and outputs are corresponding SPARQL queries with natural language placeholders for IDs.

- **LLM Role:** As in the direct approach, this finetuned model also generates queries with placeholders, which are then resolved by the same post-processing script.
- **Contribution to Problem Solution:** Finetuning is intended to improve the LLM's understanding of SPARQL query patterns and Wikidata's specific conventions, potentially leading to more accurate and robust query generation.

4.3 Gemini (MCP)

This approach integrates the Model Context Protocol (MCP) to enable dynamic interaction with Wikidata APIs.

- **MCP Tools:** The LLM is provided with two tools: '`wikidata search`' to search for Wikidata entities by name, and '`query wikidata`' to execute a SPARQL query against the Wikidata endpoint.
- **Iterative Refinement:** The LLM's system instruction guides it through an iterative process:
    1. Use 'wikidata search' to look up relevant entities to find their IDs and understand their data structure.
    2. Construct a SPARQL query using the retrieved information.
    3. Use 'query wikidata' to execute the generated query and inspect the results.
    4. If the generated query was malformed or the results are not sensible, the LLM revises its query and re-tests.
- **Contribution to Problem Solution:** Allowing the LLM to perform its own entity matching prevents semantically correctly interpreted entities from being matched with the wrong IDs. Its ability to interpret the entity data structure allows it to formulate more precise relational queries. Real-time query testing enables the LLM to refine its queries based on actual database responses.

4.4 MCP (Finetuned)

This approach combines the finetuned model with the dynamic capabilities of MCP.

- **Finetuned MCP Model:** This variant uses the same finetuned Gemini 2.5 Flash model as in approach No. 2, but it is also equipped with the MCP tools and follows the iterative refinement process described in approach No. 3.
- **Contribution to Problem Solution:** This solution aims to leverage both pre-existing knowledge and dynamic interaction for optimal query generation, combining the learned patterns from finetuning with the real-time contextual awareness and self-correction provided by MCP.

5 Evaluation

To assess the performance of the different LLM approaches, a benchmark dataset of natural language queries with their corresponding correct SPARQL queries was used. The dataset was designed to test various aspects of query generation, including complex constraints, multi-hop relationships, and logical operations. For each prompt in the benchmark, the generated query was executed, and its number of results was determined, along with its overlap with the benchmark query results. For each model, the following metrics were calculated to evaluate the quality of the generated SPARQL queries:

5.1 Metrics

- **Average Jaccard Index:** The size of the intersection of results from the generated query and the benchmark query is divided by the size of their union. This is repeated for all queries in the benchmark, and the average is taken.
- **Global Jaccard Index:** For the set containing all results from all generated queries and the set containing all results from all benchmark queries, the size of the intersection is divided by the size of the union.
- **F1-macro Score:** This is the average of the F1 score for each result.
$$F1 = \frac{2 \times \text{True Positives (TP)}}{2 \times \text{TP} + \text{False Positives} + \text{False Negatives}}$$
- **F1-micro Score:** This calculates the F1 score globally by counting the total true positives, false negatives, and false positives.

The Average Jaccard Index and the F1-macro score should be considered more meaningful measures of performance for grading these results than the Global Jaccard Index or the F1-micro Score, as they treat all results equally regardless of size, while the latter two give higher weight to queries with more results.

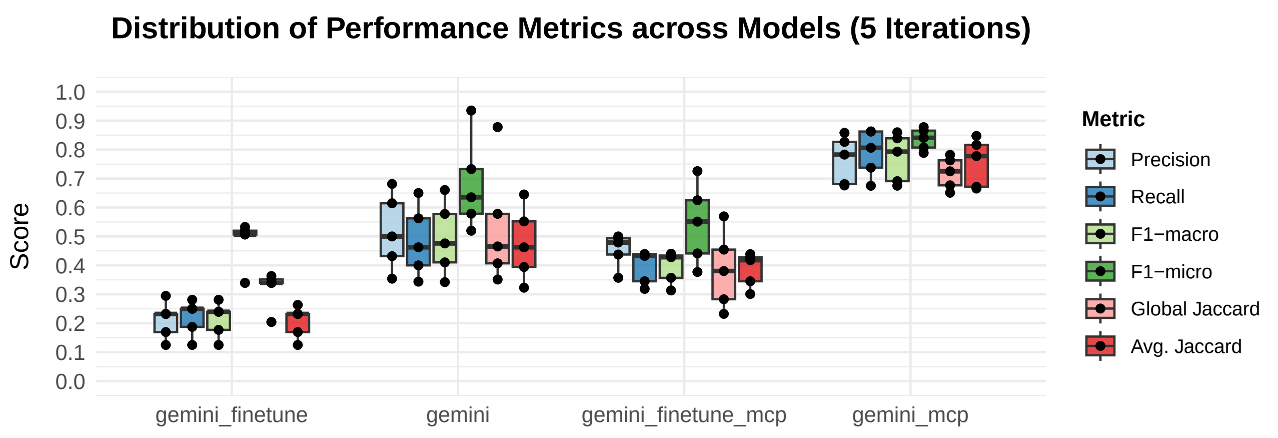

Figure 1 Measuring model performance on different metrics

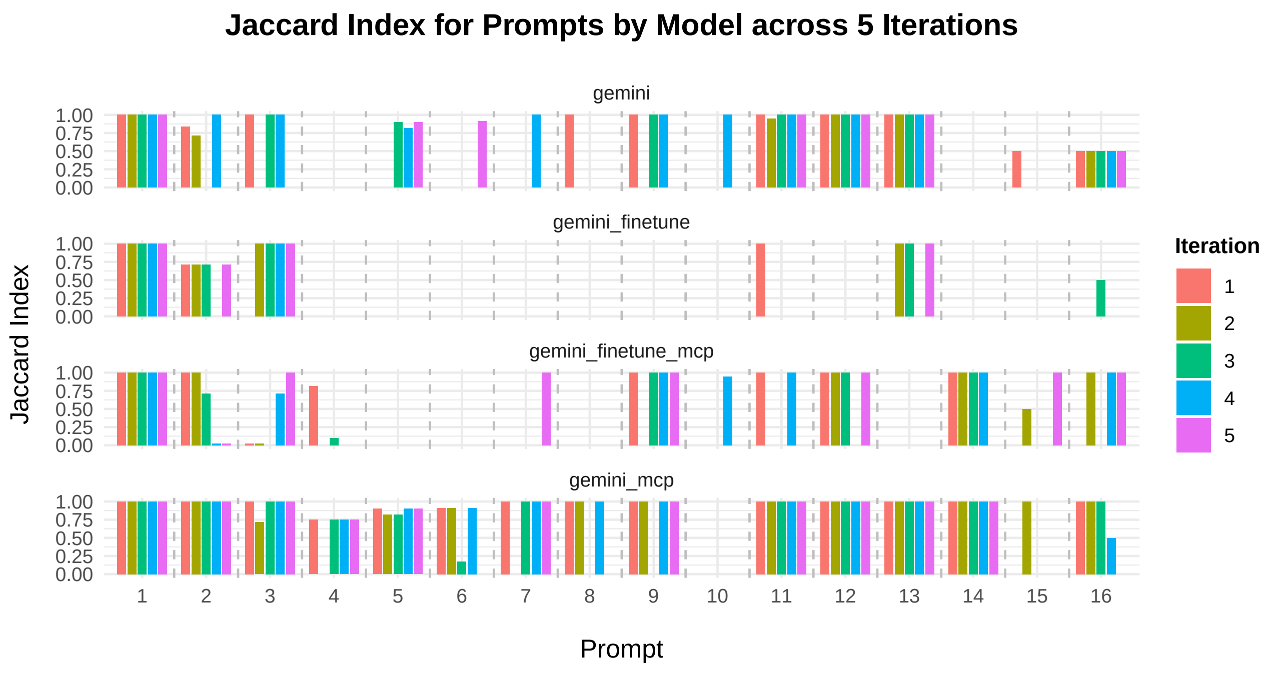

Figure 2 Jaccard Index for Prompts by Model across 5 Iterations.

5.2 Main Results

The evaluation revealed distinct performance characteristics for each approach:

5.2.1 Gemini 2.5 Flash (Direct):

This model often struggled with precise entity and property matching, as its reliance on simple string replacement for placeholders could lead to incorrect ID mappings if the search API returned an irrelevant top result. For example, the query asking for "US presidency" start dates would fail because the entity "US presidency" does not exist; the dates are saved in individual president entries.

5.2.2 Gemini (Finetuned):

Surprisingly, this model was often less capable than its plain counterpart. This is possibly because the inherent pre-training of the LLM already encompassed SPARQL and other query languages, meaning that finetuning provided little benefit. The specific finetuning data, which used natural language placeholders without prefixes, may have inadvertently hindered its ability to generate queries with the required prefixes.

5.2.3 Gemini (MCP):

This approach demonstrated significant improvement, especially for queries requiring precise entity resolution and iterative refinement. It proved to be the most performant and consistent across all considered metrics, being statistically significantly more performant than any other model ($p < 0.005$ for both average jaccard index and F1 macro score).

5.2.4 MCP (Finetuned):

This option generally underperformed, probably because it is supposed to create real SPARQL queries but was trained on queries with natural language placeholders in them, leading to more frequent syntactical errors. If such a mistake occurs, when using the 'query wikidata' tool to test its queries, it will be told why the query failed and provided with a real SPARQL query to remind it what the syntax should look like. Even this measure, which drastically improves performance, is insufficient to nullify the negative effects of learning on queries which are syntactically different from the output it is expected to generate.

5.3 Beyond the Main Metrics

No model response ever delivered entirely imprecise results, meaning that if a query yielded no matches, it was because the query genuinely had no answers. This is a strong indication that the semantic gap issue, where a model fundamentally misunderstands the user's intent, is not currently a bottleneck in LLM natural language to SPARQL translation.

Response time is a critical performance metric in practical applications. While finetuned models generally exhibit a slightly increased generation latency compared to their baseline counterparts, this latency is still lower than that of the MCP variants, a consequence of the iterative nature of the MCP process. This increase in latency is not large enough to impede their feasibility in real-world scenarios, but is an important factor to acknowledge nevertheless.

The raw result data from all tests can be found alongside the project's source code on GitHub[2].

6 Conclusions and Future Work

6.1 Lessons Learned

The most significant lesson from this project is the critical importance of providing LLMs with real-time access to external tools and information. The Model Context Protocol (MCP) dramatically improved the LLM's ability to generate accurate SPARQL queries by enabling entity disambiguation and query validation against the actual Wikidata API. This iterative feedback loop proved far more effective than simple post-processing. Despite these advancements, LLMs continue to struggle with highly complex logical reasoning, particularly involving intricate relationships or multi-conditional filtering that demand a deep understanding of data structures. Finetuning did not provide tangible benefits and, in many cases, seemed to hurt performance, possibly due to a suboptimal training data format.

6.2 Future Work

Future research could explore:

- **Advanced Query Optimization:** Techniques for the LLM to generate more efficient SPARQL queries, especially for large result sets or complex joins. This might involve teaching the LLM about SPARQL query plan optimization or providing tools to estimate query cost.
- **Better-suited Finetuning Datasets:** Generating a purpose-built dataset that includes natural language placeholders with prefixes could significantly increase the performance of the non-MCP finetuned model, since it would match the exact format of the queries it is supposed to generate.
- **User Feedback Integration:** Incorporating a mechanism for users to provide direct feedback on generated queries and results. This human-in-the-loop approach could further improve the application's performance and identify persistent weaknesses.
- **Explainable AI (XAI):** Making the LLM's query generation process more transparent by explaining why a particular SPARQL query was chosen or how certain entities were resolved. This would help understand why suboptimal or malformed queries were returned, enabling more precise prompt engineering.

This project represents a promising step toward making knowledge bases more accessible. While significant progress has been made with MCP-enabled LLMs, continued research into more robust reasoning, error handling, and user interaction is crucial for widespread adoption. Even currently, Gemini 2.5 Flash, when given the ability to search for entities and test generated queries, is capable enough for non-SPARQL proficient users to generate most queries or for SPARQL-proficient users to significantly speed up complex query creation.

References

[1] Wikidata:Data access. https://www.wikidata.org/wiki/Wikidata:Data_access. Accessed: July 4, 2025.

[2] Xaver Baumgartner and Yun Zhou. GitHub: Wikiquery. https://github.com/XaverBaumgartner/Wikiquery_TUM. Accessed: July 9, 2025.

[3] Mohnish Dubey, Debayan Banerjee, Abdelrahman Abdelkawi, and Jens Lehmann. Lc-quad 2.0: A large dataset for complex question answering over wikidata and dbpedia. In *Proceedings of the 18th International Semantic Web Conference (ISWC)*. Springer, 2019.

[4] Tim Schwabe, Louisa Siebel, Patrik Valach, and Maribel Acosta. Q-NL Verifier: Leveraging Synthetic Data for Robust Knowledge Graph Question Answering. *arXiv preprint arXiv:2503.01385*, 2025.