NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCES
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

BSc THESIS

Advancing GeoSPARQL Query Generation on YAGO2Geo: Leveraging Large Language Models and Automated URI Injection from Natural Language Questions

Efstratios - Paschalis D. Kakalis

Supervisors: **Manolis Koubarakis,** Professor
Sergios-Anestis Kefalidis, Associate Researcher

ATHENS

OCTOBER 2024

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Προώθηση της Δημιουργίας GeoSPARQL Ερωτημάτων στο YAGO2Geo: Αξιοποίηση Μεγάλων Γλωσσικών Μοντέλων και Αυτοματοποιημένης Εισαγωγής URI από Ερωτήσεις σε Φυσική Γλώσσα

Ευστράτιος - Πασχάλης Δ. Κακαλής

Επιβλέποντες: **Μανώλης Κουμπαράκης,** Καθηγητής
Σέργιος - Ανέστης Κεφαλίδης, Συνεργαζόμενος Ερευνητής

ΑΘΗΝΑ

ΟΚΤΩΒΡΙΟΣ 2024

BSc THESIS

Advancing GeoSPARQL Query Generation on YAGO2Geo: Leveraging Large Language Models and Automated URI Injection from Natural Language Questions

Efstratios - Paschalis D. Kakalis
S.N.: 1115201900068

SUPERVISORS: **Manolis Koubarakis,** Professor
Sergios-Anestis Kefalidis, Associate Researcher

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Προώθηση της Δημιουργίας GeoSPARQL Ερωτημάτων στο YAGO2Geo: Αξιοποίηση Μεγάλων Γλωσσικών Μοντέλων και Αυτοματοποιημένης Εισαγωγής URI από Ερωτήσεις σε Φυσική Γλώσσα

Ευστράτιος - Πασχάλης Δ. Κακαλής
Α.Μ.: 1115201900068

ΕΠΙΒΛΕΠΟΝΤΕΣ: **Μανώλης Κουμπαράκης,** Καθηγητής
Σέργιος - Ανέστης Κεφαλίδης, Συνεργαζόμενος Ερευνητής

ABSTRACT

Question Answering (QA) over Knowledge Bases, particularly Knowledge Graphs (KGs), has become an essential task in Natural Language Processing. This task allows users to retrieve precise information from structured datasets by asking questions in natural language. However, querying KGs typically requires the use of complex query languages like SPARQL, which necessitates a deep understanding of both the KG's structure and ontology. For non-expert users, generating accurate queries in these technical formats can be highly challenging. To make Knowledge Graphs more accessible, it is essential to develop interfaces that allow users to interact with KGs through simple, natural language questions, without needing to understand SPARQL. In this study, we build on this concept by addressing the challenge of geospatial Question Answering, specifically focusing on generating GeoSPARQL queries that correspond to any given natural language question.

This thesis investigates the development of an end-to-end system for generating GeoSPARQL queries from natural language inputs leveraging LLMs. This study targets the YAGO2geo Knowledge Graph. The motivation behind this thesis is that traditional methods for query generation struggle with fixed vocabularies and complex KG structures, particularly for geospatial data. To address these challenges, this work focuses on leveraging open-source LLMs, with a particular emphasis on the Mistral 7B model, and introduces novel *URI-injection* techniques to enhance the accuracy and efficiency of SPARQL query generation.

The study evaluates several state-of-the-art (SOTA) LLMs of various sizes, comparing open-source models to proprietary models. Through named-entity disambiguation, fine-tuning, and prompt engineering, the thesis demonstrates how injecting relevant URIs during query generation can significantly improve model performance, particularly in cases where knowledge about specific entities is sparse. We produce a fine-tuned Mistral model, trained on a carefully processed train set, that shows substantial improvements in query accuracy, outperforming larger, more resource-intensive models.

The ma in contributions of this work are:

- **A systematic evaluation of existing LLMs for SPARQL query generation:** We will conduct three distinct evaluations and present them in detail, drawing conclusions on the current capabilities of various LLMs in this specific task. Our findings are verified by cross-examining the results from multiple evaluation metrics.
- **A novel prompt-engineering framework for geospatial question-answering:** ”URI-injection,” designed to enhance LLM performance in SPARQL query generation without expensive fine-tuning, making it versatile and easy to apply across multiple tasks and ontologies.
- **A fine-tuned and quantized Mistral v0.2 7b model**, matching the state-of-the-art accuracy on the GeoQuestions1089 dataset, while maintaining computational efficiency through 4-bit precision.

SUBJECT AREA: Natural Language Processing, Large Language Models

KEYWORDS: NLP, LLM, KG, Question Answering, Artificial Intelligence, NED

ΠΕΡΙΛΗΨΗ

Πρόσφατα επιτεύγματα των Μεγάλων Γλωσσικών Μοντέλων (ΜΓΜ) σε διάφορα προβλήματα Επεξεργασίας Φυσικής Γλώσσας (ΕΦΓ) επιτρέπουν την δημιουργία καινοτόμων συστημάτων ερωταπαντήσεων που βασίζονται σε Γνωσιακά Διαγράμματα (ΓΔ), αξιοποιώντας τη δύναμη των ΜΓΜ. Στόχος είναι η δημιουργία ενός συστήματος που θα μετατρέπει φυσική γλώσσα σε GeoSPARQL ερωτήματα που την απαντούν, χρησιμοποιώντας ένα ανοιχτού-κώδικα ΜΓΜ. Ωστόσο, προκύπτουν πολλές προκλήσεις, καθώς η παραγωγή SPARQL ερωτημάτων αποτελεί δύσκολο έργο ακόμη και για προηγμένα μοντέλα τελευταίας τεχνολογίας.

Η μελέτη επικεντρώνεται στο ΓΔ YAGO2Geo και χρησιμοποιεί το μεγαλύτερο σχετικό σύνολο δεδομένων, το GeoQuestions1089. Υπάρχουσες μέθοδοι για την παραγωγή ερωτημάτων αδυνατούν να διαχειριστούν μεγάλο πλήθος ερωτήσεων λόγο σταθερών λεξιλογίων και πολύπλοκων δομών ΓΔ, ιδιαίτερα όσον αφορά τα γεωχωρικά δεδομένα. Για την αντιμετώπιση αυτών των προκλήσεων, η εργασία επικεντρώνεται στη χρήση ΜΓΜ, με ιδιαίτερη έμφαση στο μοντέλο Mistral 7B, και εισάγει μια καινοτόμο τεχνική *εισαγωγής URIs* για την αύξηση της ακρίβειας και αποδοτικότητας στην παραγωγή SPARQL ερωτημάτων.

Η μελέτη αξιολογεί αρκετά μοντέλα κορυφαίας τεχνολογίας όπως το Mistral, το Llama και το GPT-4o. Μέσω της αποσαφήνισης οντοτήτων, της προσαρμοσμένης εκπαίδευσης (fine-tuning) και της μηχανικής προτροπών (prompt engineering), η εργασία δείχνει πώς η εισαγωγή σχετικών URIs κατά τη διάρκεια της παραγωγής ερωτημάτων μπορεί να βελτιώσει σημαντικά την απόδοση των μοντέλων, ιδιαίτερα σε περιπτώσεις όπου η γνώση για συγκεκριμένες οντότητες είναι περιορισμένη. Τα προσαρμοσμένα μοντέλα, εκπαιδευμένα σε ένα προσεκτικά επεξεργασμένο σύνολο δεδομένων, παρουσιάζουν σημαντικές βελτιώσεις στην ακρίβεια των ερωτημάτων, υπερβαίνοντας μεγαλύτερα και πιο απαιτητικά μοντέλα, και καθιερώνουν ένα νέο σημείο αναφοράς για την παραγωγή GeoSPARQL ερωτημάτων έναντι του YAGO2Geo ΓΔ. Επιπλέον, επισημαίνονται οι τομείς στους οποίους τα τρέχοντα μοντέλα δεν ανταποκρίνονται πλήρως, υποδεικνύοντας μελλοντικές ερευνητικές κατευθύνσεις, όπως η προσαρμοσμένη εκπαίδευση για τη βελτιωμένη αυτόματη δημιουργία URIs.

Οι κύριες συνεισφορές αυτής της εργασίας είναι:

- **Μια διεξοδική ανάλυση δημοφιλών ΜΓΜ στην παραγωγή GeoSPARQL**, χρησιμοποιώντας τρεις διαφορετικές μεθόδους αξιολόγησης, οι οποίες θα επιβεβαιωθούν μέσω διασταυρωμένης εξέτασης (cross-examination).
- **Ένα καινοτόμο πλαίσιο μηχανικής προτροπών**, η "Εισαγωγή URIs", σχεδιασμένο για να ενισχύει την απόδοση των ΜΓΜ στην παραγωγή SPARQL ερωτημάτων χωρίς την ανάγκη δαπανηρής εκπαίδευσης.
- **Ένα προσαρμοσμένο και κβαντισμένο μοντέλο Mistral v0.2 7b**, που θέτει νέο σημείο αναφοράς στην ακρίβεια στο σύνολο δεδομένων GeoQuestions1089, διατηρώντας παράλληλα την αποδοτικότητα μέσω 4-bit ακρίβειας.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Επεξεργασία Φυσικής Γλώσσας, Γλωσσικά Μοντέλα

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: ΕΦΓ, ΜΓΜ, ΓΔ, Συστήματα ερωτωαπαντήσεων, ΤΝ

Στους γονείς μου και τον αδερφό μου, που πάντα με στηρίζουν.

CONTENTS

1. INTRODUCTION 13

2. BACKGROUND AND RELATED WORK 16

3. MODEL EVALUATION 18

3.1 LLM Selection 18

3.2 Empirical LLM Evaluation 18

3.3 Target Dataset Creation 22

3.4 LLM Evaluation Based On Query Similarity 22

3.5 Result-based LLM Evaluation. 25

3.6 Evaluation Method Comparison 25

3.7 Instructions For Optimal Llama 3 Usage 30

3.8 Summary 30

4. PROMPT ENGINEERING FOR IMPROVED REASONING 32

4.1 Chain-of-Thought Prompting 32

4.2 Results and Analysis of CoT Prompting 32

5. INFORMATION AUGMENTED GENERATION 34

5.1 Named Entity Recognition and Disambiguation 34

5.2 Previous Work on YAGO2geo Entity Linking 36

5.3 LLM-powered NER and NED. 36

5.4 Results and Evaluation of the URI-Generating Methods 39

5.5 System Limitations and Error Analysis 42

5.6 Evaluation of SPARQL query generation with URI injection. 45

5.7 Conclusion 47

6. FINE-TUNING THE SPARQL QUERY GENERATION MODEL 48

6.1 Data Preparation 48

6.2 Parameter-Efficient Fine-Tuning (PEFT) Techniques 48

6.3 Experiments 49

6.4 Results 51

7. CONCLUSIONS AND FUTURE WORK **55**

7.1 EVALUATION OF OUR SYSTEMS 55

7.2 SUMMARY AND FUTURE WORK 55

ABBREVIATIONS - ACRONYMS **57**

APPENDICES **58**

A. PROMPTS USED IN THIS WORK **59**

A.1 PROMPTS USED DURING EMPIRICAL EVALUATION 59

A.2 PROMPTS USED DURING STRING SIMILARITY AND RESULTS-BASED EVALUATION . . 61

A.3 CHAIN-OF-THOUGH PROMPTS USED 63

REFERENCES **70**

LIST OF FIGURES

3.1 Example of a question answered correctly with URI-injection. 27
3.2 Pearson and Spearman correlation coefficient between the result-based metric and the similarity-based metrics. 29

4.1 Example of a question that failed to be translated to SPARQL with standard prompting but succeeded with CoT. 33

5.1 Example function of NER and NED systems. 35
5.2 LLM-IUG learning curves. 40
5.3 Incorrect concept identification due to the concept being part of an instance. 43
5.4 Incorrect concept identification due to lack of ontology knowledge and reasoning. 43

6.1 Learning Curves for the Norm-Simple-Trunc-8 model. 52
6.2 Example of the question-answering capabilities of the Mi-F8 model. 53
6.3 Comparison of generated and ground-truth ASK query for a given question. 53
6.4 Example of the question-answering capabilities of the Mi-F8 model. 54

LIST OF TABLES

3.1 Table of examined LLMs and their respective size (in billions). 19
3.2 Scores of each model on all metrics. 20
3.3 Final OSGen and InUK scores of all models. 21
3.4 Model and method performance based on query similarity using BLEU and our proposed HybridBLEU metric. 24
3.5 Result-based accuracy of the experiments from Table 3.4. 26
3.6 Result-based accuracy of each model under best performing methods. . . . 27
3.7 Alternative versions of the best performing models that show poor performance. 28
3.8 Performance of SOTA GPT models based on query similarity and query results. 29

4.1 Chain-of-Thought vs plain few-shot learning accuracy. 33

5.1 Entity linking methods alongside their true positive and false positive percentages. 40
5.2 Concept Identifiers and their respective TP and FP percentages. 40
5.3 SPARQL generation accuracy with the current best performing method and automatic URI injection methods. 46
5.4 SPARQL generation accuracy with GPT 4o-mini and URI-injection methods. 47

6.1 Resulting accuracy of fine-tuned Mistral models on the full test set and a simpler subset lacking rdfs ontologies. 51

7.1 Comparison of accuracy scores between GeoQA2 and our methods. 55

PREFACE

This thesis is part of my undergraduate studies at the Department of Informatics and Telecommunications, National and Kapodistrian University of Athens. It represents my final requirement before completing my bachelor's degree. I would like to express my gratitude to my supervisors, Professor Manolis Koumparakis and Sergios-Anestis Kefalidis, for their unwavering support throughout all stages of this work. I would also like to acknowledge Sergios' valuable contributions, particularly during the ideation phase.

1. INTRODUCTION

Question Answering is a computer science discipline within the fields of information retrieval and natural language processing that is concerned with building systems that automatically answer questions that are posed by humans in a natural language. These systems typically analyze and interpret natural language questions to retrieve relevant information, often from structured databases, knowledge graphs, or unstructured text. The goal is to provide accurate, concise responses based on the input question.

The challenge of question answering becomes even more pronounced when users pose questions with a geospatial dimension, such as “Which rivers cross London?”, “Is there a Levi’s store in Hannover?”, or ”Are any countries bordering Greece in the EU, having joined after 1997, but don’t use the Euro as their currency?”. Answering such questions is crucial, especially when geospatial information is combined with thematic data like population statistics, environmental factors, or economic indicators. This integration allows for deeper analysis, offering insights into how spatial relationships influence specific themes. For instance, understanding how population density overlaps with flood-prone areas can enhance disaster preparedness, while combining economic and geographic data can drive regional development. This makes geospatial question answering an invaluable tool for tackling complex, real-world issues across various sectors. [26] highlights this by showcasing a digital assistant’s value in three key use cases: vessel detection for maritime surveillance, monitoring water bodies dynamics (e.g., floods), and constructing training datasets for AI models. The assistant simplifies the process of answering geospatial questions using satellite data , serving both experts and non-experts.

However, answering geospatial questions requires structured data enriched with spatial information, typically stored in geospatial knowledge graphs (e.g., YAGO2 [15], YAGO2geo [19], WorldKG [9], and KnowWhereGraph [17]) or linked geospatial data. Traditionally, the standard way to retrieve knowledge from geospatial KGs or RDF stores is through the SPARQL query language and its geospatial extensions such as GeoSPARQL [29] and stSPARQL [22].

Unfortunately, the requirement of SPARQL knowledge reduces the accessibility of KGs for non-technical users. Enabling non-technical users to pose geospatial questions in natural language introduces several additional complexities. The system must be able to identify and interpret the spatial representation of entities, such as points or polygons, based on the context of the question. The interpretation of spatial operations and relationships can also vary depending on the scale or scope of the query. For instance, the term ”near” is understood differently when comparing questions like “Which countries are near Greece?” and “Which POIs are near the Acropolis?”. Furthermore, the system needs to account for the variability in spatial language, such as differentiating between ”north of Greece” and ”northern Greece.” Finally, certain questions may require complex geometric calculations, which can be computationally demanding.

Another challenge in generating SPARQL queries is the need to ground the queries in the target KG using Uniform Resource Identifiers (URIs), which represent abstract or physical resources, like places or concepts. Every resource has a unique URI that must be known to reference it in a query. This becomes difficult because URIs often have irregular names that don’t follow predictable patterns. Therefore, a GeoSPARQL generation system must not only understand the GeoSPARQL language and natural language nuances but also consistently retrieve or infer the necessary URIs for accurate query generation.

To meet these challenges, it is crucial to develop robust, user-friendly systems that seamlessly convert natural language queries into valid geospatial queries, utilizing the power of Knowledge Graphs and advanced QA techniques.

For this purpose, many techniques have already been used. One popular method is the use of neural machine translation (NMT) [45] which usually involves an encoder-decoder architecture, where input sentences pass through the encoder to generate a vector that holds semantic information, then the decoder produces tokens at each time step based on the encoder's output and previous tokens. However, this method has a few drawbacks. First, these models have fixed vocabularies, which will not cover all KGs, URIs and extensions of SPARQL such as GeoSPARQL. Furthermore, training such models is very difficult as you will need a very large model with a huge relative training corpus since the problem at hand is very complex [8].

The recent advancements in generative Large Language Models have led to remarkable breakthroughs in a wide array of Natural Language Processing tasks, ranging from language translation [49] to text summarization [24] and even code generation [6]. Their ability to model complex linguistic structures and perform tasks across multiple domains has sparked considerable interest in extending their capabilities to more specialized fields. One such area is the automatic generation of SPARQL queries from natural language prompts, which has significant potential for improving accessibility to structured data in Knowledge Graphs and other semantic data repositories [36], [25], [23], [32], [5], [34].

At first glance, it might seem intuitive that the same success achieved in other NLP tasks would carry over to SPARQL query generation. LLMs have already demonstrated a high degree of proficiency in generating structured outputs like code, SQL [50], or mathematical formulas [27] from natural language instructions, and SPARQL query generation would appear to follow the same logic. However, many challenges specific to SPARQL query generation remain unresolved, and the task proves to be significantly more complex than it initially appears.

One major issue LLMs face in this context is hallucination. LLMs often generate inaccurate or entirely fictitious URIs, a problem attributed to their reliance on parametric memory [48]. While LLMs excel at storing vast amounts of information learned from training data, their internal representations may not capture the precise knowledge needed to generate exact URIs, especially when dealing with highly specialized knowledge domains such as those found in geospatial or domain-specific Knowledge Graphs (e.g., YAGO2geo). The models are prone to substituting missing information with plausible but incorrect alternatives, leading to invalid or inaccurate SPARQL queries. This poses a significant challenge when the goal is not just to generate syntactically correct queries but to ensure their semantic accuracy by linking them to the right entities and relationships in the Knowledge Graph.

Another limitation stems from the logical reasoning required to generate complex SPARQL queries. While LLMs can often learn the basic syntax and structure of SPARQL queries, they frequently lack the deep reasoning abilities needed to construct queries that correctly capture the intent behind a natural language prompt. This is especially true when the query involves intricate relationships, filters, or advanced geospatial operations. In these cases, the model might produce a syntactically correct SPARQL query that fails to retrieve the desired results because it misinterprets the question or incorrectly structures the query logic. We will document this behaviour in detail.

Furthermore, the variability in natural language expressions adds another layer of complexity. Different phrasings of a question can change the structure or focus of a query, and the model must not only comprehend the surface meaning of the question but also translate that understanding into precise SPARQL operations. This requires both a deep understanding of the domain and an ability to reason through multiple linguistic and contextual variations.

Despite these challenges, the potential for LLMs to revolutionize SPARQL query generation remains highly promising. To address these issues, we propose investigating a combination of techniques aimed at improving accuracy and reliability. Specifically, we will explore fine-tuning models on domain-specific datasets to enhance their understanding of GeoSPARQL and the associated ontologies. Additionally, we will introduce and evaluate a prompt-engineering method called URI-injection which aids the query generation process by supplying the model with the necessary URIs. As LLMs continue to advance, these strategies will contribute to overcoming existing limitations, making natural language-to-SPARQL query generation more practical and robust.

In this study, we will compare a wide range of state-of-the-art LLMs on the task of SPARQL query generation from natural language prompts. We will evaluate these models against a carefully selected set of handcrafted prompts and provide empirical metrics to assess their performance. Our goal is to identify the best small-scale, open-source LLM (limited to models that can run on 16GBs of VRAM) as a baseline for SPARQL generation tasks. To provide further context, we will also compare these results to GPT-4, a large-scale model with allegedly 1.76 trillion parameters by OpenAI [30], and assess how the smaller LLMs measure up. Building on these findings, we will then develop a complete system for generating SPARQL queries from natural language prompts, leveraging the best-performing models. This system will incorporate advanced techniques such as prompt engineering, information-augmented prompting through named entity recognition and disambiguation, and fine-tuning on the available training corpus for the YAGO2geo KG.

All of the files relative to this work are included in the thesis' GitHub page [1] and the best performing fine-tuned model weights are openly available on Hugging Face through the Transformers library [2].

The remainder of this thesis is structured as follows. Chapter 2 presents a review of related works and the current state of geospatial question answering systems. Chapter 3 provides a systematic evaluation of existing models, identifying the most suitable approaches for geospatial question answering tasks. Chapter 4 explores the Chain-of-Thought framework, examining its impact on enhancing model reasoning capabilities. Chapter 5 investigates the URI-injection method, applying it to several models and assessing the resulting improvements. Chapter 6 discusses common fine-tuning techniques and presents QLoRA fine-tuning experiments conducted on a selected model, along with an evaluation of the findings. Finally, Chapter 7 concludes the thesis by discussing the results of the research and suggesting directions for future work in the field of geospatial question answering.

---

[1] https://github.com/StratosKakalis/Thesis

[2] https://huggingface.co/Stratos-Kakalis/norm_trunc_no_rdfs_8_epoch

2. BACKGROUND AND RELATED WORK

The field of geospatial question answering has experienced significant growth in recent years. One good example of this work is GeoQA2 [34], the most recent version of the GeoQA family of engines [33] . GeoQA2 is a geospatial question-answering system designed to handle complex geographic queries by leveraging traditional NLP methods. The system integrates various components, including a dependency parse tree generator, a concept and instance identifier, a geospatial relation identifier, a property identifier, and a query generator. The dependency parse tree generator performs part of speech tagging. Based on these generated tags the system targets certain words and checks if they belong to existing libraries of URIs using string similarity measures. This system also makes use of TagMeDisambiguate [11], an entity linker system that accepts string input and links resources with their respective URIs. These components work together to analyze user input, identify relevant geographic features, and generate SPARQL queries based on the YAGO2geo ontology. This structured approach enables GeoQA2 to efficiently handle complex questions by breaking them down using linguistic and geospatial processing techniques. GeoQA2 is of significant importance for this work as it targets the same underlying Knowledge Base, YAGO2geo, and also serves as the baseline of our research.

Ruled-based approaches likes GeoQA2 have a number of inherent limitations, including a finite dictionary of predefined terms and templates and a lack of understanding when it comes to vague or ambiguous natural language. These constraints hinder their flexibility in dealing with more complex or nuanced queries, making it difficult to scale or generalize across varied geospatial questions.

To attempt and mitigate some of these issues we turn to several recent approaches for generating SPARQL queries from natural language queries based on neural architectures. [41] designed a sequence-to-sequence system that utilizes bi-directional LSTM for generating SPARQL templates. However, this translational approach cannot handle out-of-vocabulary tokens, due to a fixed tokenizer vocabulary. [37] presents SGPT, an approach using a stack of Transformer encoders to embed linguistic features from natural language questions, as well as entity and relation information, to the GPT-2 model. While entities and relations representations are fed to the model in SGPT, providing their connections in the underlying KG is missing. Thus, generating correct triple sequences in the final SPARQL queries is error prone due to unknown graph structures. Another strong approach was introduced in [4], where authors improve on the state of the art KGQA by training the T5 model to generate skeleton SPARQL queries and truncated KG embeddings, that are used to fetch candidate entities for the skeleton query. Finally, [23] presented SPARQLGEN, a powerfull one-shot prompting approach for generating SPARQL queries. Unlike other methods, SPARQLGEN requires no additional training or fine-tuning for the specific task.

Building on these works, we will explore LLM-powered geospatial QA systems with a focus on the YAGO2geo knowledge graph, similar to the approach taken by GeoQA2.

We will leverage prompting techniques like zero-shot, few-shot, and Chain-of-Thought (CoT) prompting [46]. Zero-shot prompting assesses the model's ability to generate SPARQL queries without examples, while few-shot provides a handful of examples to guide the model. Chain-of-Thought prompting breaks complex tasks into intermediate steps, improving reasoning accuracy. These methods will be explored to enhance natural language-to-SPARQL query generation.

The LLMs relevant to this task are general-purpose pre-trained models, as no SPARQL-specific models currently exist. We focus on small-scale models that can be trained on limited hardware and provide fast inference, analyzing various popular options listed in Table 3.1. As an exception, we will also evaluate GPT 4, which cannot run on our hardware, to compare the performance of small-scale models against that of a massive model.

This work makes use of the GeoQuestions1089 dataset, introduced in [21] and updated in [20]. It comprises of 1089 questions, their corresponding GeoSPARQL queries and their answers over the geospatial knowledge graph YAGO2geo. It is the largest dataset of its kind available for geospatial question answering tasks. We are interested in the YAGO2geo KG for various reasons. It is based on the subset of YAGO2 which includes only geoentities i.e., entities that have latitude/longitude co-ordinates associated with them (presumably, representing their center). YAGO2geo enriches the geospatial dimension of some of these geoentities with detailed geometries, namely lines, polygons and multi-polygons taken from official administrative datasets (for Greece, the United Kingdom and the Republic of Ireland) and the Global Administrative Areas dataset. Hence, YAGO2geo can be used to answer questions that could not be answered by YAGO2 because detailed administrative geospatial knowledge is required (e.g., “Which counties of England are crossed by river Thames?”). Additionally, for natural features such as lakes and rivers, the respective YAGO2 geoentities are enriched with detailed geometries from OpenStreetMap. Finally, YAGO2geo includes new geontities present in the above administrative datasets and OSM that were not present in YAGO2. YAGO2geo currently contains 703 thousand polygons and 3.8 million lines. YAGO2geo represents geographic knowledge by using the YAGO2 ontology, the GeoSPARQL ontology and the ontologies especially developed by the YAGO2geo developers for each dataset mentioned above. [21]

3. MODEL EVALUATION

In this chapter, we conduct a comprehensive evaluation of our candidate LLMs. The primary objective is to assess their performance in generating GeoSPARQL queries and to determine the most suitable pre-trained model that will serve as our baseline for subsequent experiments.

3.1 LLM Selection

For this study we selected strictly open-source LLMs in the spirit of open collaboration and transparency. Open-source language models encourage community involvement, peer review, and accessibility of code and resources. Furthermore, due to hardware limitations, we will examine 'small-scale' LLMs. Specifically, models that required less than 16GBs of VRAM to run. This would typically limit us to models with less than 7B parameters, however with the use of quantization for lower precision we will be able to load much larger models of up to 13B parameters. Any observed loss in precision due to quantization will also be discussed in this thesis.

The selected LLMs are all open-source and comply with the memory limitations, furthermore all of these models were chosen based on their performance including meta's Llama 3 model which was released during the creation of this work and provides new SOTA performance across various benchmarks. We also test the proprietary model GPT-4 by OpenAI. This model is massive (estimated 1.76 trillion parameters [30]), thus we expect much better performance from it, however it will be informative to showcase how the smaller open-source models will compare. The complete list of the LLMs used in this study is seen in Table 3.1.

3.2 Empirical LLM Evaluation

We will start our approach to evaluating and choosing the proper LLM based on an empirical evaluation of a few different metrics. Namely, we will test our models on two proposed categories: **Overall SPARQL Generation (OSGen)** and **Inherent URI Knowledge (InUK).** Each category has several metrics, for OSGen we score the model based on its ability to:

- *M1.*) Generate queries (correct or wrong).
- *M2.*) Not generate repeating words or phrases.
- *M3.*) Not hallucinate (specifically SPARQL keywords, not necessarily URIs).
- *M4.*) Generate syntactically correct SPARQL queries.

For InUK we score the model based on its ability to:

- *M5.*) Generate valid URIs (based on their total count).
- *M6.*) Generate valid wikidata URIs.

Model	# of Parameters
Llama 2 [42]	7B
Llama 2 (finetuned for text-to-SQL)[42]	7B
Llama 2 [42]	13B (Quantized to 6-bit precision)
CodeLlama [38]	13B (Quantized to 6-bit precision)
Mistral v0.2 [18]	7B
Mistral v0.2 Instruction Tuned [18]	7B
Dolphin mistral v2.8 [18]	7B
Hermes 2 Pro mistral [18]	7B
Google T5 [35]	3B
Google T5 v1.1 xl [35]	3B
Falcon Instruct [2]	7B
Llama 3 [10]	8B (Quantized to 8-bit precision)
Llama 3 Instruction Tuned [10]	8B (Quantized to 8-bit precision)
Google Gemma [28]	2B
Google Gemma Instruction Tuned [28]	2B
GPT-4 [30]	1.76T (estimated)
GPT-4o [30]	-
GPT-4o mini [30]	-

Table 3.1: Table of examined LLMs and their respective size (in billions).

- *M7.*) Generate valid DBpedia URIs.
- *M8.*) Generate valid YAGO URIs.

Each model's OSGen score will indicate how familiar the model is with SPARQL itself and it will serve as a good indication of the logical reasoning capabilities of the LLM that is required to form complex SPARQL queries. On the other hand the InUK score will appraise the pre-trained model's embedded knowledge regarding various available KGs such as: wikidata [44], DBpedia [3] and YAGO.

It is obvious that each metric has a different amount of importance. We translate that importance into the final score by empirically setting weights according to the expected significance of each metric. For instance, the *M4* metric for the OSGen score is much more indicative of a model's capabilities than the *M1* metric (which may reward a model even for incorrect queries).

The formula for $\text{OSGen}(\text{model})$ is:

$$\text{OSGen}(\text{model}) = \frac{0.4 \times \text{model}_{\text{M1}} + 0.6 \times \text{model}_{\text{M2}} + 0.7 \times \text{model}_{\text{M3}} + 1 \times \text{model}_{\text{M4}}}{0.4 + 0.6 + 0.7 + 1},$$

where model_{Mi}, for i in $\{1, 2, 3, 4\}$ are the scores of $model$ in each of the metrics: *M1, M2, M3, M4.*

Similarly, the formula for $\text{InUK}(\text{model})$ is:

$$\text{InUK}(\text{model}) = \frac{0.5 \times \text{model}_{\text{M5}} + 0.6 \times \text{model}_{\text{M6}} + 0.6 \times \text{model}_{\text{M7}} + 0.8 \times \text{model}_{\text{M8}}}{0.5 + 0.6 + 0.6 + 0.8},$$

where model_{Mi}, for i in $\{5, 6, 7, 8\}$ are the scores of $model$ in each of the metrics: *M5, M6, M7, M8.*

Model	M1	M2	M3	M4	M5	M6	M7	M8
Llama-2-7b	0.929	0.929	0.857	0.231	0.379	0.500	0.600	0.000
Llama-2-7b SQL-tuned	1.000	0.857	0.857	0.571	0.588	0.571	0.818	0.286
Llama 13b (6-bit)	0.929	0.143	0.500	0.462	0.452	0.625	0.500	0.500
CodeLlama 13b (6-bit)	1.000	0.214	0.929	0.500	0.548	0.545	0.818	0.250
Mistral 7b v0.2	1.000	0.643	1.000	0.714	0.679	0.727	0.857	0.500
Mistral 7B v0.2 Instruct	1.000	0.643	0.929	0.286	0.509	0.385	0.636	0.500
Dolphin mistral v2.8	0.857	0.857	0.929	0.167	0.378	0.444	0.273	0.000
Hermes 2 Pro mistral	1.000	0.786	0.786	0.357	0.509	0.643	0.429	0.167
Google T5-3b	0.286	0.000	0.071	0.000	0.467	0.000	0.500	0.429
Google T5v1.1-xl	0.357	0.000	0.000	0.000	0.556	1.000	1.000	0.000
Falcon-7b-Instruct	0.786	0.929	0.857	0.455	0.556	0.750	0.500	0.143
Llama 3-8b (8 bit)	1.00	0.86	1.00	0.36	0.57	0.64	0.60	0.33
Llama 3-8b -it (8 bit)	1.00	0.93	0.93	0.57	0.59	0.60	0.75	0.33
Gemma 2b	0.79	0.50	1.00	0.83	0.66	0.80	0.89	0.50
Gemma 2b-it	1.00	0.71	1.00	0.57	0.56	0.64	0.83	0.67
GPT-4-1.76t	1.00	1.00	1.00	0.93	0.76	0.89	0.80	0.30

Table 3.2: Scores of each model on all metrics.

The test prompts were hand crafted to test each model in all of the above metrics. There are 14 prompts, for this experiment, and they test SPARQL query generation with zero-shot, one-shot and few-shot learning. Furthermore, some prompts specify the KG to be used by the model and the required generated queries vary in complexity.

After running inference on all models and scoring each model in each metric by hand we received the metric performance scores on Table 3.2, we then calculate the OSGen and InUK scores in Table 3.3.

In Table 3.2, we can draw a few conclusions from the combined performances of all the models on the metrics. Specifically:

- It is already obvious that GPT-4 is superior to the rest of the models. This is expected due to the massive size of the model.
- Models struggle to form syntactically correct SPARQL. Hallucinations and repeating words or phrases are very prominent.
- Models struggle with the YAGO KG, presumably, because there are not as many datasets that involve the use of YAGO, thus it appears rarely in the models' training corpus.
- Models also struggle with Wikidata URI generation, partly due to the fact that Wikidata URIs contain arithmetic codes which are impossible to guess and, thus, difficult to independently infer.
- Models tend to perform best at DBpedia, since it's a well known KG and it has a simpler URI structure so models can often correctly guess the link to a certain entity by applying reasoning (e.g. 'Mavro' corresponds to 'http://dbpedia.org/resource/Mavro').

In Table 3.3, from the final OSGen and InUK scores we draw these conclusions:

Model	OSGen	InUK
Llama-2-7b	0.652	0.340
Llama-2-7b SQL-tuned	0.772	0.543
Llama 13b (6-bit)	0.470	0.520
CodeLlama 13b (6-bit)	0.622	0.517
Mistral 7b v0.2	0.815	0.676
Mistral 7B v0.2 Instruct	0.638	0.507
Dolphin mistral v2.8	0.620	0.248
Hermes 2 Pro mistral	0.659	0.412
Google T5-3b	0.061	0.350
Google T5v1.1-xl	0.053	0.591
Falcon-7b-Instruct	0.713	0.457
Llama 3-8b (8 bit)	0.730	0.518
Llama 3-8b -it (8 bit)	0.807	0.550
Gemma 2b	0.795	0.696
Gemma 2b-it	0.778	0.680
GPT-4-1.76t	**0.974**	**0.717**

Table 3.3: Final OSGen and InUK scores of all models.

- Our previous observation regarding GPT-4 is validated, as it continues to show superior results across both metrics. However, despite the smaller size of open-source models, the performance gap is not as wide, suggesting that the difference in capabilities could potentially be bridged with effective fine-tuning.
- Some models, like Google T5, struggle with SPARQL query generation, as expected. Google specifies that these models aren't pretrained on a large corpus, necessitating substantial fine-tuning. Given our focus on pre-trained models that require minimal tuning for SPARQL, T5 is not suitable.
- Llama 2 presents a solid baseline with respectable performance, which is further enhanced in its SQL-finetuned version.
- Interestingly, the 13b version of Llama 2 under-performs compared to the 7b version. This may be due to factors like differing training data, reasoning capabilities, or potential loss of precision during quantization. Perhaps, a different prompt, better aligned with the 13b model's training corpus could improve results.
- The Mistral models, along with the Falcon and Google Gemma models, performed impressively well, consistently surpassing Llama 2.
- Llama 3, released during this study, stands as the new SOTA in many NLP tasks, offering a clear advantage over its predecessor.

In conclusion, Llama 3 and Mistral v0.2 stand out among the rest. Furthermore, Google Gemma 2b models appear very powerful as they both had exceptional performance considering their much smaller size. In certain cases were hardware or speed is limited, Gemma could prove more useful. Another notable performance comes from Falcon-7b-Instruct.

3.3 Target Dataset Creation

Moving forward we will leverage the information we have gathered so far to build a model for the specific task of GeoSPARQL generation against the YAGO2geo knowledge graph. To accomplish this, we require a dataset comprising question/query pairs with URIs extracted from the target knowledge graph. Therefore, we gathered data from two distinct sources: firstly, the GeoQuestions1089 dataset introduced by [21], and secondly, the GeoQuestions201 dataset introduced by [33].

Upon collecting both datasets, we merged them and filtered out three invalid question/query pairs lacking queries. Additionally, some queries provided by the authors exhibited minor variations in question format without introducing any semantic difference. Presumably, this was intended to assess the models' proficiency in understanding language even under unconventional structures. However, considering the inherent versatility of LLMs in natural language understanding, we opted to remove these repetitive question/query pairs from the dataset. Their inclusion could potentially bias the models during training by presenting redundant questions and it would make for invalid training if some queries existed on the training and test sets.

The resulting dataset has a total of 1214 question/query pairs and will be referred to as GeoQuestions1214.

3.4 LLM Evaluation Based On Query Similarity

After narrowing the list of potential LLM candidates down to three prominent models, Mistral, Llama-3 and Gemma, we proceed with certifiable experiments designed to yield quantitative and reliable results. The first set of tests will focus the evaluation on the generated queries by similarity to the target queries.

During these experiments, we will introduce new methods to enhance the models' performance, such as incorporating URI data within the prompts, prompting the models with GeoSPARQL relations, and fine-tuning the models.

First of all, let's review the evaluation method. For these first tests, we will need an automatic evaluation metric. Since our models rarely produce valid queries, it would not be wise to evaluate them based on the query results. For now, it's preferable to evaluate the models on the similarity between the generated queries and the ground truth queries from the GeoQuestions1214 dataset.

To assess the best metric for this similarity evaluation, three metrics stand out:

- **Modified Jaccard Similarity**: Typically, this metric does not consider token frequency or order. However, with a slight modification, it can be adapted to account for token frequency, which is crucial for addressing issues such as models mistakenly repeating certain keywords. This adjustment ensures that models are not rewarded for redundant repetitions. The metric is simple to interpret and decreases quickly when the token sequences do not align.
- **Cosine Similarity**: This metric is useful for comparing the similarity of documents based on their token vectors. For SPARQL queries, cosine similarity is less punitive than other metrics, which is beneficial when the model struggles with the basic structure of the query. Instead of receiving a very low score each time, the model is

rewarded the more it resembles the original. However, this metric does not account for token order.

- **BLEU Score**: Designed for comparing sequences of words, BLEU takes into account both the order of tokens and their frequency. It is primarily used for evaluating machine translation tasks, which closely relates to the task of SPARQL query generation.

It is evident that the BLEU score is the most suitable metric for this task, and it has also been used in related works. However, it would be prudent to provide both the BLEU score and a hybrid score incorporating all three metrics for each model. Evaluating the differences in scores can be insightful, and employing multiple metrics may help mitigate some of BLEU's discrepancies.

Moving forward, the two metrics provided for every model we will evaluate are **BLEU** and **HybridBLEU**, defined as follows:

$$\begin{aligned}\text{HybridBLEU} = 0.20 \times \text{Jaccard Similarity} + 0.20 \times \text{Cosine Similarity}\\ + 0.05 \times \text{Parsability Reward} + 0.55 \times \text{BLEU Score}\end{aligned} \tag{3.1}$$

The weights in HybridBLEU were chosen empirically, based on the anticipated impact of each metric in the final similarity score. By combining these metrics, HybridBLEU aims to provide a comprehensive evaluation that balances different aspects of similarity and functionality.

Due to computational limitations, we will use a random subset of the GeoQuestions1214 dataset with 200 questions. This is because we plan to evaluate each model several times with various techniques to help improve it's generation abilities. Presumably, 200 randomly selected questions will not cause the resulting evaluation to deviate significantly from a test on the complete dataset. Each experiment, with a combination of different prompting methods is listed in Table 3.4. The detailed list of prompts can be found in Appendix A.2. In each row, the model is followed by tags representing these methods:

- **0-s:** This tag means zero-shot learning. The model was not shown any examples of what it is supposed to generate.
- **URI-i:** This tag means gold-standard URI-injection. We extracted the URIs from the dataset's ground-truth queries and supplied them in the prompt along with the natural language question.
- **Geo-i**: This tag means that a GeoSPARQL ontology description was provided in the model's prompt. Presumably, the model isn't familiar with GeoSPARQL. This is an attempt at using heterogeneous data through in-context learning instead of training which embeds the knowledge into the model's parametric memory.
- **few-s**: This tag means few-shot learning. The model was shown several examples of the queries it's meant to generate. The number in the parentheses next to this tag indicates the number of examples in the prompt.

The goal of these experiments is to assess the best combination of model and methods. The two models that we will test are the two most promising models from our earlier experiments on Subsection 3.2, Mistral v0.2 7b and Llama 3-8b Instruct (8 bit).

Model and methods	BLEU	HybridBLEU
Mistral v0.2 7b: 0-s	0.135	0.213
Mistral v0.2 7b: 0-s, URI-i	0.175	0.303
Mistral v0.2 7b: 0-s, URI-i, Geo-i	0.111	0.260
Mistral v0.2 7b: few-s(3), URI-i	0.393	0.532
Mistral v0.2 7b: few-s(6), URI-i	0.443	0.572
Mistral v0.2 7b: few-s(3)	0.225	0.364
Llama 3 8b Instruct (8 bit): few-s(3)	0.150	0.238
Gemma 2b: few-s(3)	0.172	0.229

Table 3.4: Model and method performance based on query similarity using BLEU and our proposed HybridBLEU metric.

The results of our experiments are presented in Table 3.4. Notably, there are significantly more tests conducted with Mistral v0.2 compared to Llama 3-it, attributable to computational constraints. On average, Mistral required approximately 40 minutes to complete each test, whereas Llama 3-it took over 4 hours per test. Due to this disparity in performance, a direct comparison between Mistral and Llama was conducted in a few-shot learning experiment. The results, illustrated in Table 3.4, clearly demonstrate that Mistral outperforms Llama, particularly when considering the substantial difference in processing time.

Despite Llama 3 being a more recent, larger, and more complex model than Mistral v0.2 , it under-performs Mistral. Presumably, this is due to the fact that:

- Mistral may have seen more SPARQL related literature in its training corpus than Llama.
- Llama was quantized to 8 bits of precision which may have resulted in some amount of loss in the quality of it's predictions [14].

Llama 3 as an off-the-shelf pretrained model performs impressively in many leaderboards, however in the huggingface open llm leader-board we can see that various models compete for the top spot with several models based on Mistral and Llama 3 [1]. This means that when the model is finetuned for a specific task, they perform rather similarly.

Perhaps, if one had the resources to test and fine-tune Llama 3-8b Instruct without any loss in precision (32 or 16 bit floats) then it would outperform Mistral. However, considering our current limitations and the specificity of our task, it is wiser to proceed with Mistral as it shows more promising results and faster compute time.

Finally, we also tested Google Gemma 2b-it on the same parameters. It is a smaller 2b-parameter instruction tuned model that performed rather impressively considering its small size. Inference was, as expected, really fast and this model would be best utilized in scenarios were processing power is limited. In our case we can afford the computational cost of Mistral which offers slightly improved performance so we will continue with it as our target model.

---

[1] https://huggingface.co/spaces/open-llm-leaderboard/open_llm_leaderboard

3.5 Result-based LLM Evaluation.

For the final and most practical evaluation, we will assess the models' accuracy in generating SPARQL queries. Specifically, we will measure the percentage of queries that the models can successfully generate and obtain valid responses from the YAGO2geo knowledge graph.

The first step in this process is to setup an endpoint that contains the YAGO2geo knowledge graph. We did this by installing GraphDB [12], a Semantic Graph Database, compliant with W3C Standards. GraphDB ran on a Linux server with 128GBs of RAM. We loaded the target KG on it and successfully set up an endpoint to which we send a get request containing the GeoSPARQL query that we want to execute and receive the retrieved results.

It is important to note that during the dataset creation process we merged two separate datasets, the GeoQuestions1089 and GeoQuestions201. The former was designed to work with the YAGO2geo knowledge graph, while the latter had additional dependencies on prefixes such as OpenStreetMap, making it impossible to infer on our endpoint. Consequently, we created a new dataset consisting solely of the GeoQuestions1089 dataset. For performance reasons, the resulting dataset contains only 100 randomly selected queries. Furthermore, some queries are very slow to execute. To counteract this performance issue, we used the GOST API [2] to materialize the SPARQL queries, thereby significantly improving inference times from the endpoint.

The next step in completing the evaluation is to compare the results between the ground-truth queries from our new dataset and the results from the generated queries. This comparison, however, is not trivial. While we can compare the results from the generated queries to the ground-truth queries when the generated query is not invalid, we may encounter several types of mismatches. Firstly, variable names can differ between two valid queries. Similarly, the order of the returned variables can vary without affecting the validity of the answer. Finally, it is possible that the model might return more data than necessary. This is not necessarily wrong, as questions can be vague, and some extra information does not undermine the question-answering nature of our models.

An initial attempt to account for these discrepancies involved normalizing the variable names by assigning specific serial names to the variables in each query. However, this approach did not address the issues of incorrect variable order and extra returned data. The final solution was to return the endpoint data as a CSV file. Each column in this file represents the values of a specific variable. We compare the columns between the ground truth and generated query while ignoring variable names altogether. If at least one column perfectly matches between the results then the results are considered correct. This logic is sound because our ground truth queries are concise and do not select any extra random variables, thus typically only returning a single column of values.

3.6 Evaluation Method Comparison

By now, we have performed multiple evaluations of the models combined with various optimization methods, each using different means of assessment. It is important to verify the validity of these evaluations by cross-examining them. For this reason we repeated the

---

[2] https://github.com/AI-team-UoA/GoST

Model	Result-based Accuracy
Mistral v0.2 7b: 0-s	0.00
Mistral v0.2 7b: 0-s, URI-i	0.03
Mistral v0.2 7b: 0-s, URI-i, Geo-i	0.00
Mistral v0.2 7b: few-s(3), URI-i	0.22
Mistral v0.2 7b: few-s(6), URI-i	0.25
Mistral v0.2 7b: few-s(3)	0.12
Llama 3 8b Instruct (8 bit): few-s(3)	0.04
Gemma 2b: few-s(3)	0.08

Table 3.5: Result-based accuracy of the experiments from Table 3.4.

experiments of Table 3.4 only this time the evaluation metric is the result-based accuracy of the models as proposed in the Subsection 3.5. The results of the experiments using the same methods and prompts as Table 3.4 can be seen in Table 3.5.

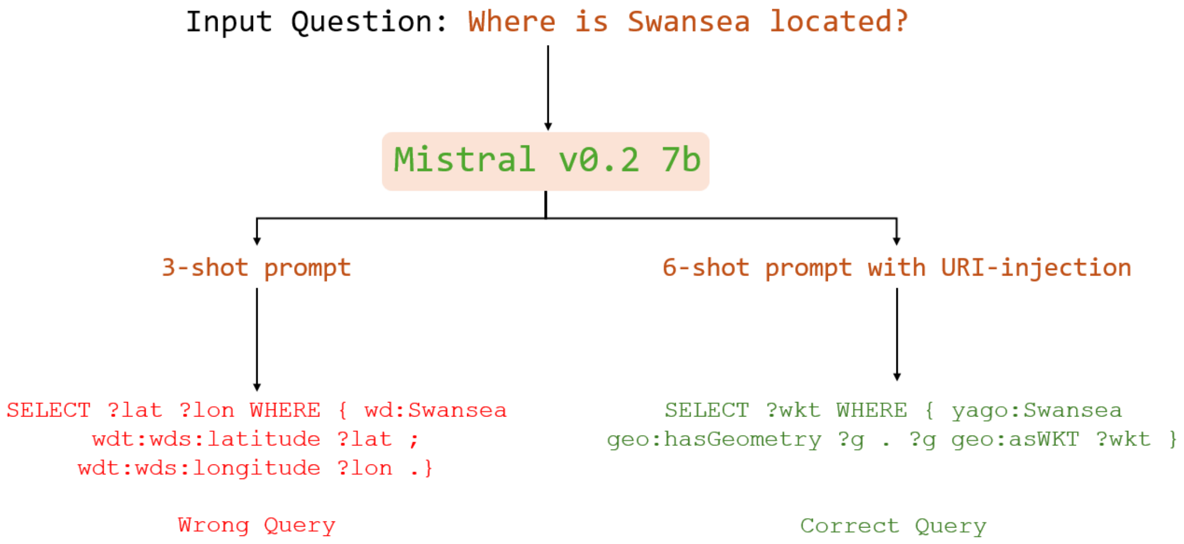
Straight away we notice a few things: the models all struggle to generate valid queries that return correct results. Specifically, two of our tests fail completely in the test set of 100 questions. Only one model is separated from the rest and that is Mistral v0.2 7b as we will explore in more detail shortly. When we supply this model with a few examples (3 and 6 in our experiments) and if we also provide the necessary gold-standard URIs for the model to create the queries then the model generates GeoSPARQL queries with 25% accuracy. In other words, one out of every four questions supplied to this pipeline leads to correct results returned from the knowledge graph.

These results clearly demonstrate that the models possess limited knowledge of SPARQL and the YAGO2Geo Knowledge Graph. Additionally, it's reasonable to assume they also lack familiarity with GeoSPARQL relations, given that GeoSPARQL and stSPARQL are an extension of SPARQL. Consequently, generating GeoSPARQL queries against the YAGO2Geo KG is beyond the initial capabilities of these models. However, by incorporating few-shot learning and heterogeneous data such as URIs in the prompts, we observe a notable improvement in accuracy. This improvement can be attributed to the models' reasoning abilities, where they correctly answer questions based on the provided context rather than relying solely on their pre-trained knowledge.

We can see an example of this behaviour in Figure 3.1. Even when provided with a 3-shot prompt that includes examples using the YAGO2Geo ontology, along with explicit instructions to follow that format, the model still opts to use Wikidata to answer the given question. This suggests that despite guidance, the model defaults to familiar data sources, possibly due to its pre-trained bias or stronger associations with Wikidata over YAGO2Geo.These tendencies are only bypassed when the model is explicitly provided with the necessary URIs, as it lacks the confidence or ability to infer them independently. By supplying the relevant URIs, the model can correctly generate the desired queries without defaulting to alternative knowledge sources.

In the experiments conducted in Table 3.5 we only have one direct comparison of all models and that is on the 3-shot learning prompt in which Mistral seems to be superior. In order to verify these finding we also compare the models on the best performing parameters, which are, 6 examples in the prompt and the URIs of the query supplied.

The findings presented in Table 3.6 reveal a significant disparity between Mistral and the other two language models, verifying the direct comparison results shown in Table 3.5 for plain 3-shot learning. These observations clearly demonstrate that Mistral is the most suit-


Input Question: Where is Swansea located?
Mistral v0.2 7b
3-shot prompt
6-shot prompt with URI-injection
Wrong Query
Correct Query

```
SELECT ?lat ?lon WHERE { wd:Swansea
wdt:wds:latitude ?lat ;
wdt:wds:longitude ?lon .}
```

```
SELECT ?wkt WHERE { yago:Swansea
geo:hasGeometry ?g . ?g geo:asWKT ?wkt }
```

Figure 3.1: Example of a question answered correctly with URI-injection.

Model	Result-based Accuracy
Mistral v0.2 7b	0.25
Llama 3 8b Instruct (8 bit)	0.17
Gemma 2b	0.11

Table 3.6: Result-based accuracy of each model under best performing methods.

Model	Alternative Model	Accu.	Alt. Accu.
Mistral v0.2 7b: few-s(6), URI-i	Mistral v0.2 7b it: few-s(6), URI-i	0.25	0.09
Llama 3 8b it (8 bit): few-s(6), URI-i	Llama 3 8b (8 bit): few-s(6), URI-i	0.17	0.05

Table 3.7: Alternative versions of the best performing models that show poor performance.

able model for our GeoSPARQL generation tasks using data from the YAGO2geo knowledge graph. This superior performance can likely be attributed to a possibly greater extent of SPARQL knowledge embedded in Mistral's training corpus.

An important distinction is that some of the examined models are instruction-tuned while others are base versions. This selection was not made at random; we chose these models based on the results presented in Table 3.2 of Subsection 3.2. However, since that evaluation was empirical and aimed at quickly narrowing down the contending models, it is prudent to compare the selected models with their alternatives, including both instruction-tuned and base versions to verify our initial findings. In our context, ”alternative models” refers to their instruction-tuned or base counterparts. For example, the instruction-tuned version of a baseline model is considered its alternative, and vice versa.

Indeed, Table 3.7 clearly demonstrates that our quickly selected models outperform their alternative versions. The performance difference is substantial, indicating that despite being based on the same model, they exhibit markedly different capabilities in SPARQL generation. Notably, the best-performing version of Llama 3 is the instruction-tuned variant, while Mistral's base version surpasses its instruction-tuned counterpart. For the instruction-tuned models simple prompting isn't optimal, thus for these models we used the chat-completions method of supplying system and user prompts. This finding suggests that for some tasks involving the execution of complex instructions, an instruction-tuned model is not necessarily the optimal choice.

By now, it is evident that none of our small-scale LLMs are performing well on this task. Therefore, we will also document the performance of the current state-of-the-art GPT-4o model and its smaller version, GPT-4o-mini. Table 3.8 presents the models and their respective result-based accuracy. We tested GPT-4o-mini using both the gold-standard URI-injection method and the plain few-shot method, while GPT-4o was tested only with the latter to evaluate the best-performing unassisted accuracy of any model. We will refer to the gold-standard URI-injection as 'assisted' URI-injection throughout this work, emphasizing that it relies on knowledge of the ground-truth query and is not fully autonomous. As anticipated, GPT-4o performs slightly better, though this comes with a significantly higher cost compared to the mini version. For each test, the similarity-based scores are also reported and these will be utilized in subsequent analyses within this subsection.

Our assisted URI-injection method provides the model with the ground-truth URIs in the exact order they appear in the target query, which may lead the model to simply ”fill in” the necessary SPARQL components without truly engaging in the task. In subsequent tests in this study we will test URI-injection methods that don't rely on the ground-truth data. Those systems will generate URIs that will not necessarily follow the order or structure of the target query. Consequently, it's crucial to demonstrate that assisted URI-injection remains effective even when the provided URIs are unordered. This test aims to disprove the notion that the model may understand reasoning patterns from ordered URI-injections from the structure of the given URI triples, something that will not be available in an autonomous system.

To test this, we shuffled the URIs before passing them to the GPT 4o mini model. The

Model	Result-based Accuracy	BLEU	HybridBLEU
GPT 4o mini: few-s(6), URI-i	0.59	0.512	0.634
GPT 4o mini: few-s(6), shuffled URI-i	0.62	-	-
GPT 4o mini: few-s(3)	0.27	0.258	0.406
GPT 4o: few-s(3)	0.29	0.338	0.475

Table 3.8: Performance of SOTA GPT models based on query similarity and query results.

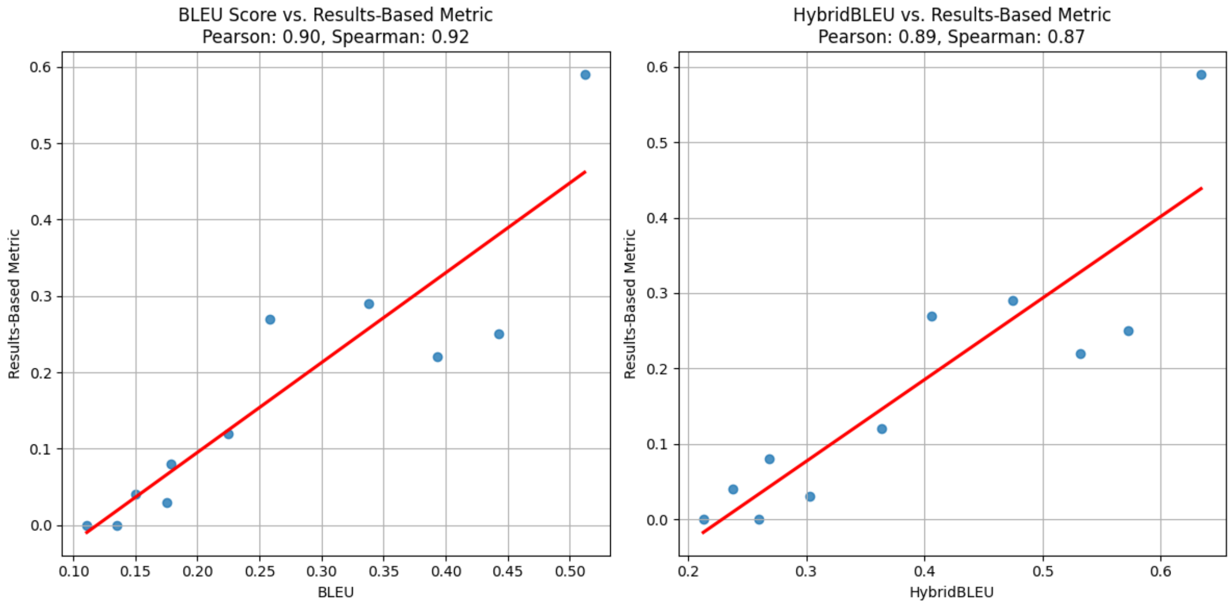


Figure 3.2: Pearson and Spearman correlation coefficient between the result-based metric and the similarity-based metrics.

results, as shown in Table 3.8, are labeled ”shuffled URI-i.” For this test, string similarity scores were excluded. Interestingly, shuffling the URIs slightly improved the model’s performance, raising its accuracy by 3%, from 59% to 62%. This improvement may be due to GPT 4o mini’s non-deterministic nature. Another hypothesis is that the shuffled URIs prompt the model to rely more on reasoning, as opposed to simply completing the SPARQL components when given ordered examples.

These results affirm the robustness of the URI-injection method, showing that when a model is provided with the correct URI, regardless of order, it performs substantially better.

To complete our cross-examination of evaluation methods, we will directly compare the results of Table 3.5 with those of Table 3.4. Specifically, we will calculate the Pearson correlation coefficient between the result-based metric in Table 3.5 and each of the similarity-based metrics in Table 3.4. We will also include the evaluation scores of the GPT models to better calculate the correlation coefficient between our metrics. The Pearson correlation coefficient, which ranges from -1 to 1, quantifies the strength and direction of the linear relationship between two quantitative variables. We suspect a linear correlation between these metrics, however, since our sample size is small we cannot draw any conclusions. Therefore, we will also compute the Spearman rank correlation. This non-parametric measure assesses the monotonic relationship between two variables.

The results presented in Figure 3.2 demonstrate the correlation between our three metrics, alongside the Pearson and Spearman correlation coefficients. Notably, when excluding the GPT models, the Pearson correlation coefficient rises to 0.97, indicating a very strong linear correlation. However, the inclusion of the GPT models, which exhibit significantly better accuracy but only moderately improved similarity scores, reduces the linear relationship and thus decreases the Pearson correlation coefficient. Despite this, we still observe two very strong positive linear correlations between the result-based metric and each of the similarity-based metrics. The Spearman correlation scores show the same strong relation between the metrics. This validation confirms the effectiveness of our similarity metrics, as their outcomes align with the actual accuracy of the models. Both similarity scores, BLEU and HybridBLEU, perform similarly, with BLEU being slightly more accurate. This suggests that HybridBLEU or other variants may not be necessary, and the most appropriate measure of query similarity for assessing a model's generation capabilities is the BLEU score. A potential improvement to HybridBLEU could involve fine-tuning its weights by framing the task as a regression problem. However, at present, this approach does not offer significant advantages and will not be pursued further.

3.7 Instructions For Optimal Llama 3 Usage

Llama 3 being a new model with SOTA performance in various NLP tasks is of great importance despite it's inferior performance in our specific task. This sections studies how to best utilize this model. During testing we noticed a peculiarity in the generation of queries with this model. All tested LLMs return the prompt provided to them followed by the generated answer. Thus, to acquire the generated query we first delete the prompt by removing a number of characters equal to the length of the prompt. However, for llama 3 this did not work because the model was not perfectly replicating the given prompt. Upon closer inspection, we discovered that Llama 3 does not allow spacing before punctuation, specifically periods and question marks, which are frequently used in SPARQL queries. This behavior likely stems from the model's training on natural language, where proper syntax dictates that no spaces should precede punctuation marks. Consequently, the model has learned to avoid leaving blank spaces before such punctuation.

Presumably, this issue can be mitigated with proper fine-tuning, however a simpler solution to this problem was comparing the prompt to the generated text character-by-character while ignoring spaces, thus finding the exact point at which the model generated the query. To find the exact query (and not include any possible explanations that the models periodically adds) we instruct all models to return the generated query within a code block denoted by triple back-ticks. On a zero shot learning prompt it would be possible to skip the entire prompt-removing process and simply get the code within the triple back-ticks but in our case we use examples that include code blocks, and the model itself might return multiple code blocks. That is why it's adamant that we properly remove the prompt to acquire the query. Finally, to resolve syntax errors caused by the lack of proper spacing within the generated SPARQL query we simply insert a space character before each period and question mark generated as long as they aren't part of a URI.

3.8 Summary

In this section, we evaluated several open-source LLMs using various methodologies. Initially, we conducted empirical assessments using a set of hand-crafted metrics designed to measure the models' ability to generate SPARQL queries. These preliminary findings helped us narrow down the list of models, enabling the subsequent automated evaluations.

To facilitate these tests, we combined the GeoQuestions1089 and GeoQuestions201 datasets, creating an appropriate benchmark for evaluation. We began by grading each model based on the string similarity between the generated queries and the ground truth. Following this, we assessed model accuracy by calculating how many correct queries were generated. This final evaluation compared the results of both generated and ground-truth queries to determine their correctness. Ultimately, we validated our evaluation methods by demonstrating their alignment and offered guidance for the optimal use of Llama 3.

4. PROMPT ENGINEERING FOR IMPROVED REASONING

The capabilities of large language models are significantly influenced by the prompts provided to them. By utilizing prompt engineering techniques, it is possible to enhance various aspects of these models, including reasoning, logic, truthfulness, optimization, and knowledge-based reasoning, without requiring extensive additional training [39]. We have already experimented with several prompt-engineering techniques, such as zero-shot and few-shot learning, yielding varying degrees of success. In this section, we will shift our focus towards prompt-engineering methods specifically designed to enhance the model's reasoning capabilities.

4.1 Chain-of-Thought Prompting

Chain-of-Thought (CoT), as introduced by [46], is an effective technique for eliciting reasoning capabilities in large language models (LLMs) without necessitating costly fine-tuning. CoT enhances LLMs' capacity to engage in complex reasoning and logical processes by generating a series of intermediate reasoning steps, commonly referred to as a ”chain-of-thought.”

There are two primary approaches to implementing CoT prompting. The first is a zero-shot method where the model is instructed with a ”Let's think step by step...” prompt. This approach encourages the model to deconstruct the main problem into smaller, more manageable sub-problems. By addressing each sub-problem individually, the model can ultimately arrive at a solution to the original task. This method mirrors human cognitive processes, where it is more intuitive to consider incremental logical steps rather than attempting to generate an immediate solution.

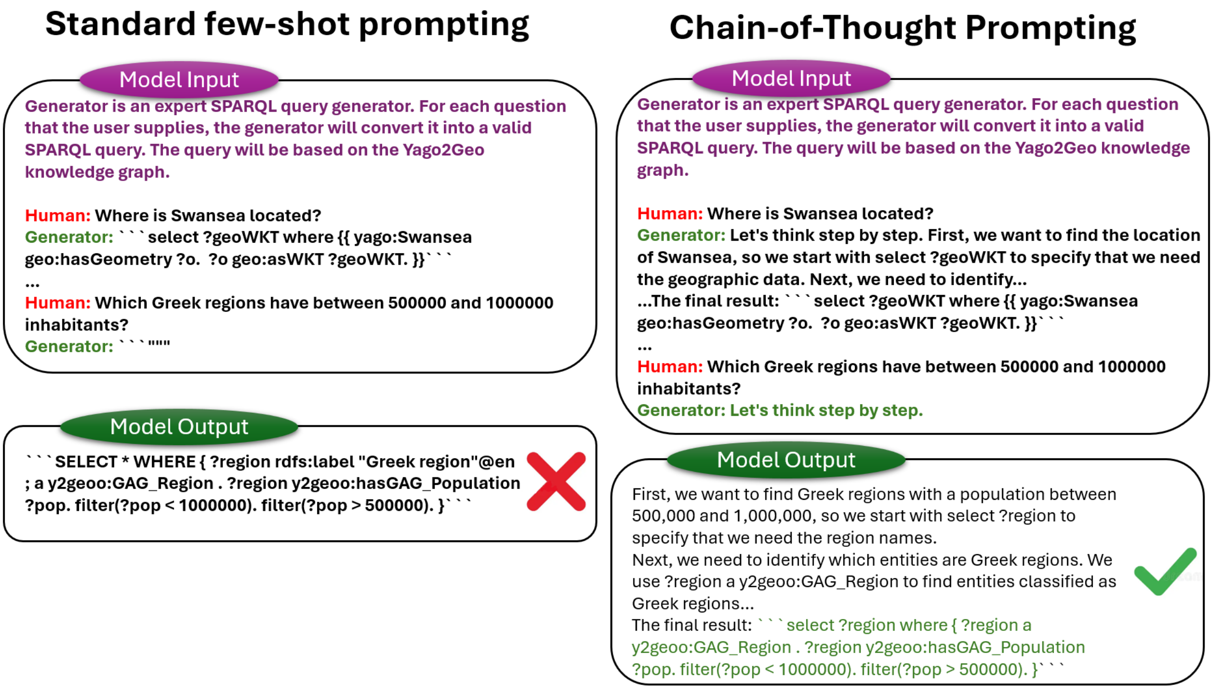
The second way to utilize CoT is to supply the model with a few examples much like few-shot learning. These examples contain the reasoning chains that are required to solve them. By examining these examples, the model gains a clearer understanding of how to perform step-by-step reasoning and can use these examples as a guide to formulate solutions for new problems.

In Figure 4.1 we can see how the model yields a correct answer only after it is prompted with the CoT method, and it formulates reasonable steps that lead to a correct SPARQL query. It's also noteworthy that the model closely follows the thought format that it was provided. Perhaps a general reasoning chain will perform best since it will apply to most questions from the task at hand.

4.2 Results and Analysis of CoT Prompting

We applied this few-shot Chain-of-Thought approach to the current pipeline, powered by the best-performing model, Mistral v0.2 7b, according to our previous tests. The goal of the following experiment is to examine how the model's performance changes from a basic 3-shot prompt and a 3-shot Chain-of-Thought prompt that contains 3 reasoning chains, one for each example.

In the first tests, we noticed a peculiar issue. The basic 3-shot prompt performed according to previous tests with an accuracy of 0.12 as seen in Table 3.5. However, the 3-shot



Standard few-shot prompting

Model Input

```
Generator is an expert SPARQL query generator. For each question
that the user supplies, the generator will convert it into a valid
SPARQL query. The query will be based on the Yago2Geo
knowledge graph.

Human: Where is Swansea located?
Generator: ```select ?geoWKT where {{ yago:Swansea
geo:hasGeometry ?o.  ?o geo:asWKT ?geoWKT. }}```
...
Human: Which Greek regions have between 500000 and 1000000
inhabitants?
Generator: ```"""
```

Model Output

~~~
```SELECT * WHERE { ?region rdfs:label "Greek region"@en
; a y2geoo:GAG_Region . ?region y2geoo:hasGAG_Population
?pop. filter(?pop < 1000000). filter(?pop > 500000). }```
~~~

Chain-of-Thought Prompting

Model Input

```
Generator is an expert SPARQL query generator. For each question
that the user supplies, the generator will convert it into a valid
SPARQL query. The query will be based on the Yago2Geo knowledge
graph.

Human: Where is Swansea located?
Generator: Let's think step by step. First, we want to find the location
of Swansea, so we start with select ?geoWKT to specify that we need
the geographic data. Next, we need to identify...
...The final result: ```select ?geoWKT where {{ yago:Swansea
geo:hasGeometry ?o.  ?o geo:asWKT ?geoWKT. }}```
...
Human: Which Greek regions have between 500000 and 1000000
inhabitants?
Generator: Let's think step by step.
```

Model Output

```
First, we want to find Greek regions with a population between
500,000 and 1,000,000, so we start with select ?region to
specify that we need the region names.
Next, we need to identify which entities are Greek regions. We
use ?region a y2geoo:GAG_Region to find entities classified as
Greek regions...
The final result: ```select ?region where { ?region a
y2geoo:GAG_Region . ?region y2geoo:hasGAG_Population
?pop. filter(?pop < 1000000). filter(?pop > 500000). }```
```

Figure 4.1: Example of a question that failed to be translated to SPARQL with standard prompting but succeeded with CoT.

Model	Few-Shot Learning	Chain-of-Thought Prompting
Mistral v0.2 7b	0.10	0.14

Table 4.1: Chain-of-Thought vs plain few-shot learning accuracy.

CoT prompt caused the model to generate only select queries, presumably because all instructions mentioned select queries so the CoT model would always create a select query as the first step of it's reasoning chain. To counteract this issue we simply substituted a random select query from the prompted examples for an ask query. For consistency we applied this change to both the CoT prompt and the basic prompt.

This led to a decrease in the base 3-shot accuracy to 0.10, which, while notable, is not unexpected. It is well-documented that altering an LLM's prompt can significantly affect its performance [40]. In this case, the simpler ”ask” query likely provided less context about the Knowledge Graph (KG) and GeoSPARQL compared to the more detailed ”select” query, giving the model less information to work with. However, this variation in performance is not necessarily a flaw in the study; rather, it highlights how sensitive models are to the prompt's structure and the examples provided.

In fact, the reduced performance underlines a key aspect of our research: the goal is not merely to supply more examples to improve accuracy but to explore how different intelligent methods, such as prompt engineering or URI-injection, can enhance model performance. This underscores that the temporary performance decline is not a setback but rather a new baseline for the performance of our models with this given prompt. The CoT model with the new examples delivered a 0.14 accuracy as seen in Table 4.1. This impressive 40% increase in performance is a result of the model's improved reasoning capabilities.

5. INFORMATION AUGMENTED GENERATION

A significant challenge in our LLM-based system stems from the inherently limited scope of knowledge contained within the model's parametric memory. This limitation is particularly evident in the model's insufficient understanding of GeoSPARQL and the YAGO2geo knowledge graph (KG). Consequently, this lack of knowledge leads to errors such as the generation of hallucinated URIs and the application of incorrect or non-existent relations, resulting in incorrect results.

To address this issue, we will explore two primary strategies for enhancing the model's knowledge base. The first strategy involves supplying the model with relevant information through prompting, which it can then utilize via ”in-context learning.” The second strategy focuses on fine-tuning the model using our existing dataset, thereby directly enriching its parametric memory.

We have already experimented with the first strategy of prompting the model with relevant URIs. As demonstrated in subsections 3.5 and 3.6 we performed this method of prompting the models with the necessary URIs to generate each query, referred to as URI-injection. We also proved that it significantly improved the performance of the system. The results indicate that LLMs effectively leverage the supplied URIs, not only by selecting the correct URIs but also by gaining a deeper understanding of the problem and generating valid query structures.

However, all previous attempts at URI-injection have been based on the ground-truth queries in our dataset. As a result, these experiments were heavily guided by the ground-truth data and do not accurately represent the system's performance when faced with novel questions.

This section focuses on generating valid URIs that are relevant to a given natural-language question, which will then be provided to the model through URI-injection. Our objective is to harness the established capabilities of URI-injection and implement them within a fully autonomous system by utilizing URIs that are generated without any external assistance.

5.1 Named Entity Recognition and Disambiguation

Named Entity Recognition (NER) and Named Entity Disambiguation (NED) are critical tasks in Natural Language Processing (NLP) that involve identifying and accurately interpreting named entities within text. These entities can belong to various categories, such as names of people, organizations, locations, and more. For our task of generating GeoSPARQL queries based on geographic questions, our primary focus is on identifying the names of locations and places, known as toponyms. After identifying these toponyms, we perform entity linking or disambiguation, which involves associating each toponym with its corresponding URI.

In Figure 5.1 we illustrate an example of a NER and NED system in action. The NER component effectively identifies toponyms within a sentence. However, since these toponyms can often correspond to multiple URIs, the NED component is crucial for linking each toponym to a specific URI. The NED system achieves this by utilizing the context provided by the original sentence to make accurate associations.

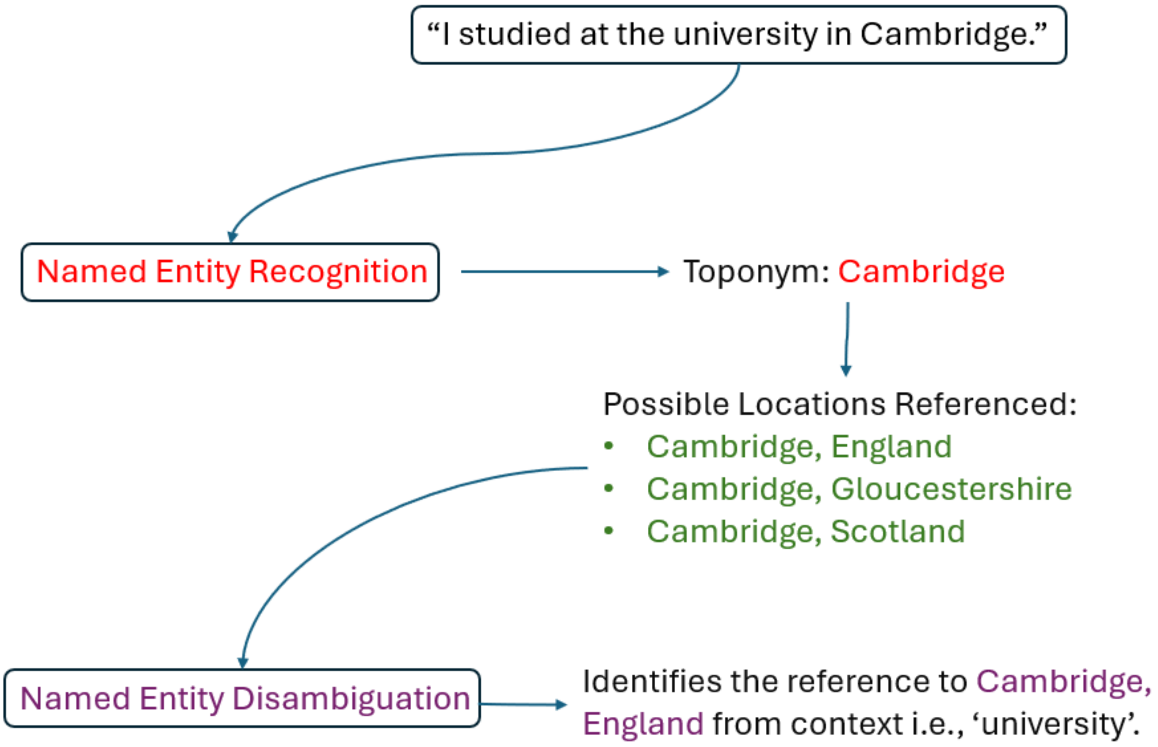

"I studied at the university in Cambridge."
Named Entity Recognition
Toponym: Cambridge
Possible Locations Referenced:
Cambridge, England
Cambridge, Gloucestershire
Cambridge, Scotland
Named Entity Disambiguation
Identifies the reference to Cambridge, England from context i.e., 'university'.

Figure 5.1: Example function of NER and NED systems.

5.2 Previous Work on YAGO2geo Entity Linking

The only two prominent question answering systems based on the YAGO2 knowledge graph are GeoQA2 [34] and the [13] GeoSPARQL translation system. This study builds on the GeoQA2 system and seeks to enhance its performance. GeoQA2 comprises multiple components, each serving distinct functions to gather the necessary information and form a valid query. One of these components is responsible for performing NER and entity linking for toponyms in the input question. We will closely examine this method and establish it as a baseline for improving entity linking performance with several alternative systems that we will propose later.

GeoQA2 handles recognition and disambiguation within a component known as the ”Instance Identifier.” This component identifies the features (instances) present in the input question, such as ”Corfu island” or ”country Mayo.” The process begins with a dependency parse tree generator that performs part-of-speech tagging and generates a dependency parse tree for the given question using the Stanford CoreNLP toolkit. The instances within the input question are identified by elements tagged as (NN, NNS, NNP) by the dependency parse tree generator. These elements are then mapped to YAGO2geo resources using TagMeDisambiguate, an entity recognition and disambiguation tool.

One important distinction to note is that while replicating GeoQA2’s code, we used WAT instead of TagMe. WAT is an entity linker hosted by the same API as TagMe, but it is preferred due to its comparable runtime performance and improved accuracy. [31]

5.3 LLM-powered NER and NED.

Large Language Models (LLMs) hold significant potential in advancing various areas of natural language processing, particularly in tasks such as NER. Leveraging this potential, we will explore several approaches, including methods that rely solely on LLMs and hybrid approaches that integrate WAT. Our goal is to evaluate these different methods to determine which one offers the best performance. The LLM we utilize is Mistral v0.2 7b Instruct, chosen due to its superior performance in our previous SPARQL query generation experiments. We opted for the Instruct version because it aligns well with the ”instruction” format required for tasks like named entity recognition and disambiguation, where the model needs to follow specific instructions closely. Unlike previous experiments, NER is a fairly straightforward task, best suited for an instruction-tuned model.

Our proposed LLM-powered entity linking systems are:

- **Naive popular URI selection (P-URI):** Using few-shot learning, we prompt our model to extract all toponyms from the input question. Once identified, we query the knowledge graph (KG) for each extracted toponym to retrieve all URI entities that contain the toponym in their name. This step is crucial because toponyms often do not perfectly match the KG’s entity names. Additionally, multiple URIs may share the same name but refer to different locations, such as Athens, Greece, and Athens, Georgia. The query returns all relevant URIs, their corresponding names, and the count of their occurrences in the knowledge graph. The P-URI method is straightforward: for each toponym recognized by the LLM, we run the query and select the most popular URI from the results, which is the entity with the highest occurrence count in the knowledge graph.

- **LLM-powered disambiguation for URI selection (LLM-ED):** This method builds on some of the functionality of P-URI. Like in P-URI, we use our LLM to extract toponyms from the input question and subsequently query the knowledge graph (KG) for each toponym to compile a list of relevant URIs, along with their names and appearance counts. However, unlike P-URI, this approach does not directly select the most popular URI. Instead, we sort the entities by popularity and select the top three results. We then prompt the model again, using one-shot learning, to perform disambiguation and identify the specific URI referenced in the input question. In this step, the model is required to demonstrate reasoning skills to accurately select the relevant URI based on the context of the question. It is worth noting that some initial experiments with this method yielded poor results because we prompted the model during the disambiguation step with all the possible URIs and asked it to return the correct one. This approach was highly prone to hallucinations. To address this issue, we revised the method by listing the URIs (ranked 1 to 3) and asked the model to return the ID of the correct URI. This adjustment significantly reduced the occurrence of hallucinations.

- **Large Language Model - Instance URI Generation (LLM-IUG):** This method introduces a different approach to collecting the target URIs by directly generating them from a prompted LLM. This is achieved by fine-tuning our LLM for the task of instance URI generation. Specifically, we applied fine-tuning to the Mistral v0.2 7b model after performing 4-bit quantization. The QLoRA fine-tuning technique, which we employed, will be discussed in depth in Section 6.2. To implement this, we first created training, validation, and test splits of our dataset, following the approach described in Chapter 6. We trained the model by supplying each question alongside the corresponding ground-truth instance URIs. For evaluation, we utilized simple zero-shot prompting to maintain consistency with the format the model was fine-tuned on.

Some useful systems used in GeoQA2:

- **Instance Identifier with the WAT entity linker (IE-WAT):** The function of the Instance identifier and the selection of the WAT entity linker have been analyzed in Section5.2. Specifically, we identify the parts of the input that are tagged as ”instances” by the parse tree generator. These instances are then processed by the WAT linker to obtain a corresponding Wikipedia link. We extract the title of the link and apply it to the Yago prefix, as this is the format used by the knowledge graph. To verify the existence of the URI, we perform a simple query against the KG, and if the URI is found, it is added to the list of relevant URIs for the question. Finally, if the prefix does not exist, we follow the same approach as in the P-URI method. We query our knowledge graph (KG) for entities that contain the toponym in their name and simply select the most popular one based on the occurrence count.

Other solutions for entity linking:

- **Instance Identifier with the WAT entity linker without Dependency Parse Tree (WAT-II):** This system operates similarly to IE-WAT but excludes the use of the Dependency Parse Tree (DPT) generator. By omitting the DPT, this approach allows for an evaluation of WAT’s inherent linking capabilities without the influence of the DPT, which could otherwise prevent certain instances from being linked. The goal is to assess whether the absence of the DPT improves or hinders entity linking performance.

- **Instance Identifier with the REL entity linker (REL-II):** This system functions identically to WAT-II, with the sole exception being the use of the REL[43] entity linker in place of WAT. This allows for a direct comparison between the two entity linking solutions, enabling an evaluation of their respective performances under identical conditions.

Based on these systems we propose the following hybrid method:

- **LLM-powered entity recognition with the WAT entity linker (LLM-WAT-ED):** This approach combines elements from our LLM-ED and IE-WAT systems. Instead of relying on a parse tree generator to identify instances in a sentence, we use our LLM for toponym recognition. Once the toponyms are identified, we employ the WAT entity linker to obtain relevant Wikipedia links for these entities, rather than querying for similarly named entities across the entire knowledge graph (KG). By leveraging the strengths of both systems, this method benefits from the enhanced accuracy of LLM-based NER and the consistent linking provided by the WAT linker.

Another useful system derived from GeoQA2 is the **Concept Identifier (CoID)**. This component is relatively straightforward, relying on two manually curated lists of classes and corresponding labels from the Yago and YAGO2geo ontologies. Each list includes all the concepts from these knowledge bases, along with descriptive names (e.g. the class YAGO2geoo:OSM bay is tagged ”Bay”). The CoID module utilizes the same dependency parse tree as the Instance Identifier component. It maps any word or sequence of words identified by the POS tags (NN, NNS, NNP, NNPS) to the ontology classes using n-grams.

The concept identifier works by iterating through the two lists of class labels from the ontology, generating n-grams (where n corresponds to the number of words in each class label), and comparing the n-grams to the respective class labels. For n-grams containing more than one word, the Jaro-Winkler string similarity measure is applied; for single-word n-grams, the Levenshtein distance is used instead.

After thorough testing, we discovered that one of the lists containing YAGO concepts was negatively impacting system performance. The issue arises from the overlap in label names between YAGO concepts and YAGO2geo concepts, even though YAGO concepts are rarely used in our dataset. As a result, the system sometimes incorrectly prefers YAGO concepts over their YAGO2geo counterparts, leading to an increase in false positives and a corresponding decrease in true positives. For this reason, CoID and other concept identifier systems will only work based on the YAGO2geo concept list.

We also evaluated the performance of the Concept Identifier without relying on the dependency parse tree, which we refer to as **Plain-CoID**. This system operates similarly to the original CoID but omits the use of POS tags. Instead, it directly employs n-grams to measure similarity between YAGO2geo concept labels and the input question. This approach simplifies the process by bypassing syntactic parsing while still aiming to identify relevant concepts through string similarity.
Based on the CoID system from GeoQA2, we propose **LLM-CoID: An LLM-powered Concept Identifier.** This system utilizes the same list of class labels from the YAGO2geo ontology. However, instead of selecting URIs based on string similarity through n-grams, we leverage our LLM to identify relevant concepts. We prompt the model with the entire

lists of class labels and instruct it to find all pertinent concepts in the input question using in-context learning. To enhance accuracy, we also employ few-shot learning.
As an alternative approach, we will attempt to generate concept URIs directly, without relying on a predefined list of class labels. This will be accomplished through fine-tuning, and we will refer to this system as **LLM-CUG** (Large Language Model - Concept URI Generation), building on the LLM-IUG framework. The system follows the same procedure, with the key difference being that it is trained to generate concept URIs instead of instance URIs.

Note that, CoID, Plain-CoID, LLM-CoID and LLM-CUG do not perform Named Entity Disambiguation (NED) as they target predefined concepts rather than instances. As a result, these systems can be effectively combined with any other NER method to enhance the overall URI generation capability of our system.

5.4 Results and Evaluation of the URI-Generating Methods

In Tables 5.1 and 5.2, we present the results of all the previously proposed methods. The results are expressed as percentages, representing the True Positive (TP) and False Positive (FP) counts for each model, ensuring a consistent scale for comparison. Alongside these metrics we also supply the precision of each method which can be viewed as the its ultimate performance metric. The TP and FP rates are supplied to explore in more detail the capabilities of each system. These percentages are calculated by measuring the total counts of Instance URIs and Concept URIs within our test set. This approach allows for a clear evaluation of each method's performance relative to the total number of relevant URIs in the test set.

It is important to clarify that the percentages for True Positives (TP) and False Positives (FP) are calculated relative to the total number of concept URIs in the test set. For instance, consider a question with 5 concept URIs, and the system generates 7 URIs, 4 of which are correct and 3 incorrect. In this case, the TP rate would be 80%, while the FP rate would be 60%. Since our systems are not restricted in the number of URIs they can generate, these percentages can exceed 100%. This happens when, for example, a low threshold for string similarity or a poorly prompted LLM generates a surplus of incorrect URIs.

Including both the True Positive and False Positive metrics is crucial for determining the optimal strategy for URI generation. Each method involves a trade-off between accuracy and precision, meaning that as TP increases, FP may also rise. Finding the right balance between these two metrics is essential. Additionally, since these URIs will be input into our LLM for SPARQL query generation in the final pipeline, we prioritize a low False Positive rate. A high False Positive rate could lead to the LLM using incorrect URIs, potentially causing hallucinations and ultimately degrading performance, possibly even performing worse than if no URI-injection was performed at all.

The results presented in Table 5.1 demonstrate that the P-URI method performs reasonably well, as the most frequently used URI in the knowledge graph is often the correct choice. However, the data also reveals that when LLM-ED is introduced, utilizing in-context reasoning, there is a slight reduction in the TP rate by 1%, but a more substantial decline in the FP rate by 10%. This suggests that the LLM Disambiguation method holds significant potential and could eventually replace the IE-WAT entity linking method.

The LLM-IUG approach, which relies solely on a large language model for URI gener-

Method	True Positives	False Positives	Precision
P-URI	36%	54%	0.40
LLM-ED	35%	44%	0.44
LLM-IUG	63%	41%	0.61
REL-II	68%	48%	0.59
WAT-II	69%	56%	0.55
IE-WAT	69%	53%	0.57
LLM-WAT-ED	67%	40%	0.63

Table 5.1: Entity linking methods alongside their true positive and false positive percentages.

Method	True Positives	False Positives	Precision
CoID	84%	30%	0.74
Plain-CoID	87%	34%	0.72
LLM-CoID	45%	50%	0.47
LLM-CUG	79%	69%	0.53

Table 5.2: Concept Identifiers and their respective TP and FP percentages.

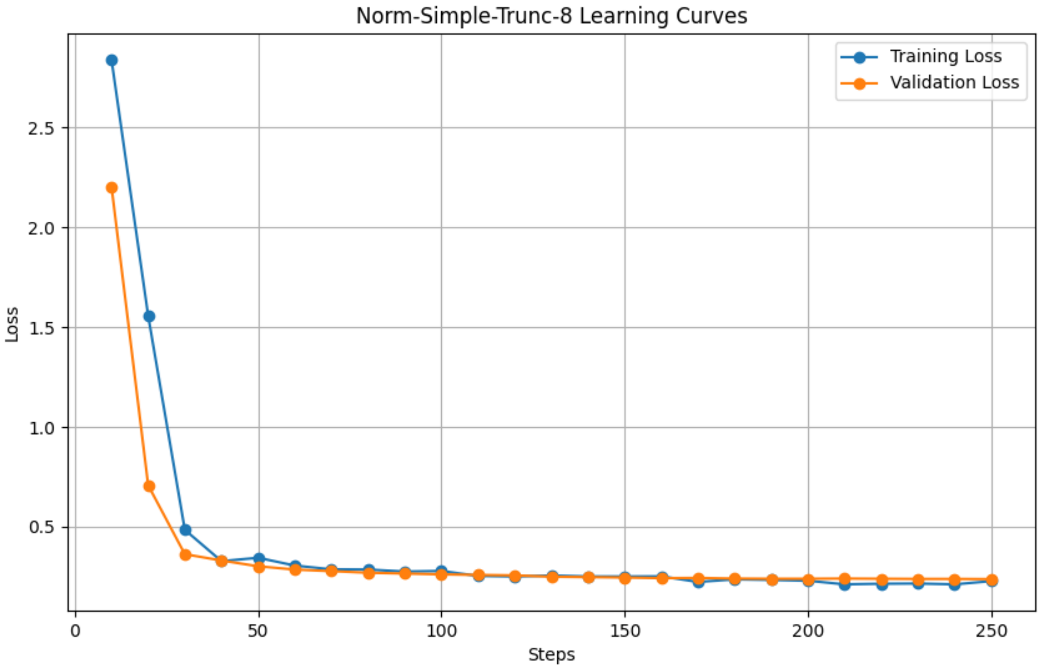


Figure 5.2: LLM-IUG learning curves.

ation, delivers impressive results by nearly doubling the true positive (TP) rate of basic LLM-based named entity recognition (NER) methods, while simultaneously reducing the false positive (FP) rate and delivering the second-best precision among all tests. This demonstrates that the model has effectively adapted to the task and learned key patterns from the underlying knowledge graph (KG). We can also verify that the model is converging well and no overfitting has occurred by examining Figure 5.2. This system is an optimal solution in scenarios where access to the KG is limited or unavailable, or when working with custom or unknown KGs for which standard entity linkers, such as REL or WAT, would not be effective. As long as we have an appropriate dataset for the related KG. Despite these constraints, the LLM-IUG system offers performance close to that of our best-performing method.

Nevertheless, at present, IE-WAT and WAT-II outperform other methods, achieving the highest observed True Positive rate of 69%, albeit with a relatively high False Positive rate of 53% and 56% respectively. REL-II offers a better balance with 0.59 precision indicating that REL might be better than WAT if used alone. Nevertheless, the hybrid approach, LLM-WAT-ED, which employs our LLM for the entity recognition phase, demonstrates a notable improvement by increasing the precision to 0.63. This suggests that LLM-WAT-ED offers the most effective balance between accuracy and precision, making it the optimal solution for our system. The LLM consistently performs accurate named entity recognition, while the WAT entity linker proves to be more robust for entity linking than relying on LLM disambiguation, which involves searching for names similar to the recognized toponyms across the entire knowledge graph. As a result, LLM-WAT-ED will be employed in our final implementation.

In addition to our NER methods, we also evaluated four Concept Identifier systems, with the results detailed in Table 5.2. This evaluation was based on the premise that concept URIs could play a crucial role during the URI-injection phase. Initially, the CoID system by GeoQA2 displayed an 84% True Positive rate and a 30% False Positive rate. These results are promising and suggest that providing the concept URIs alongside the instances will improve our query generation capability.

We tested a variant system, Plain-CoID, which operates without the Dependency Parse Tree (DPT), aiming to increase the True Positive (TP) rate. The underlying assumption was that DPT reduced the number of string similarity checks, thereby limiting opportunities to capture TPs. Plain-CoID achieved an 87% TP rate, the highest among all concept identifier methods, along with a 34% False Positive (FP) rate. As expected, both TPs and FPs increased, since the system was no longer restricted to words most likely representing concepts. However, the net improvement in TPs was only 3%, while the FPs increased by 4%. This trade-off made Plain-CoID less effective than the original CoID.

We introduced LLM-CoID to address discrepancies that will be discussed in subsection 5.5. However, this new system underperformed compared to the string similarity-based methods, achieving a True Positive (TP) rate of 45% and a False Positive (FP) rate of 50%. While these results might seem discouraging, there is potential in leveraging an LLM for this task, as it can utilize reasoning to make selections that string similarity methods cannot. For now, however, CoID remains the more effective approach.

The LLM-CUG approach showed improvement over LLM-CoID, achieving a true positive (TP) rate of 79%, comparable to the string similarity methods. However, this system also produced the highest false positive (FP) rate of 69% across all concept URI generation tests, making it unsuitable for this particular scenario despite its gains in TP accuracy.

5.5 System Limitations and Error Analysis

Upon closer inspection of the system outputs, we identified some characteristic faults specific to each system:

NER systems powered by LLMs:

- Some errors stem from the inherent nature of the entities themselves. Many entities in the knowledge graph have confusing or misleading names. For example, the correct URI for ”Dublin” is labeled as ”DUBLIN CITY COUNCIL” while another URI simply labeled ”Dublin” exists, which also has a higher occurrence count. However, this latter URI refers to a different entity and is therefore incorrect. This type of mistake is one that even a human might make, indicating that the issue may lie more with the imperfections of the knowledge graph than with the model itself.
- It is evident that the P-URI system is prone to errors because the most popular URI is not always the correct one. However, the LLM-ED system also exhibits a significant number of errors, primarily due to its insufficient disambiguation capabilities. The model often fails to apply proper reasoning, leading to obvious mistakes. Enhancing the model’s performance in disambiguation could potentially be achieved through improved prompting techniques, such as Chain-of-Thought prompting, or by employing a larger and more capable model.

GeoQA2’s Instance Identifier:

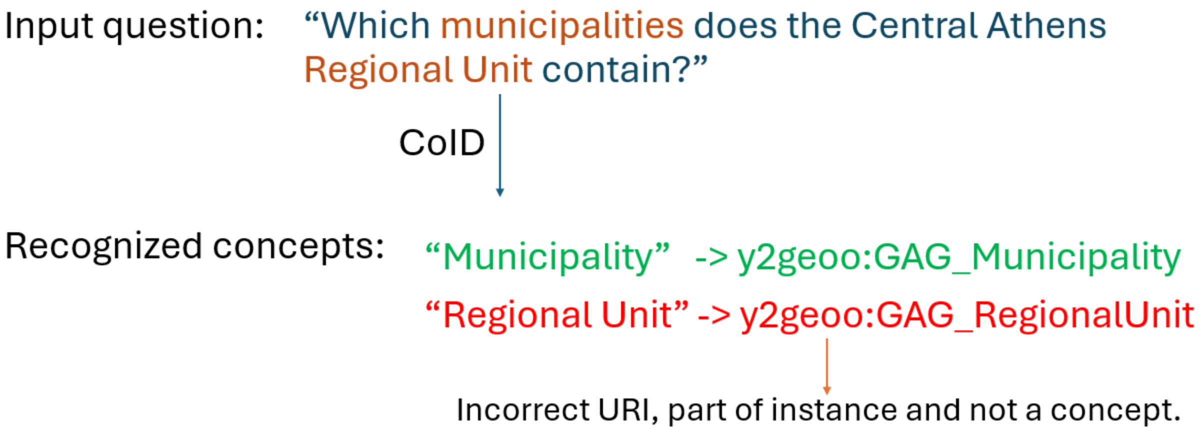
- The primary issue with this system lies in the fact that WAT is a general-purpose entity linker, not specifically optimized for toponyms. This problem is compounded by the dependency parse tree approach, which identifies instances based on broad POS tags. As a result, words that are not toponyms often get passed to the entity linker. Since WAT is designed to handle a wide range of entities, it frequently links these non-toponym words to irrelevant URIs, leading to a high number of False Positive results.

LLM-WAT-ED:

- This system effectively combines the complementary strengths of the other two approaches, leveraging the enhanced toponym-recognition capabilities of the LLM with the superior entity-linking accuracy of the WAT linker. This integration yields the best overall results. Remaining errors can largely be attributed to the inherent limitations of the current models. Future improvements, such as advanced prompting techniques and the use of a larger, more capable LLM, could potentially lead to significant enhancements in performance.

String Similarity-based Concept Identifiers:

- These systems, despite outperforming their LLM-based counterpart, exhibit notable shortcomings. With the inclusion of both YAGO2geo and YAGO classes, their performance actually falls behind that of the LLM-based approach. This issue arises due to a significant overlap between the labels in the two class lists. Fortunately, because YAGO concepts are rarely used in our dataset, this problem can be mitigated by excluding the YAGO class list entirely.



Input question: “Which municipalities does the Central Athens Regional Unit contain?”

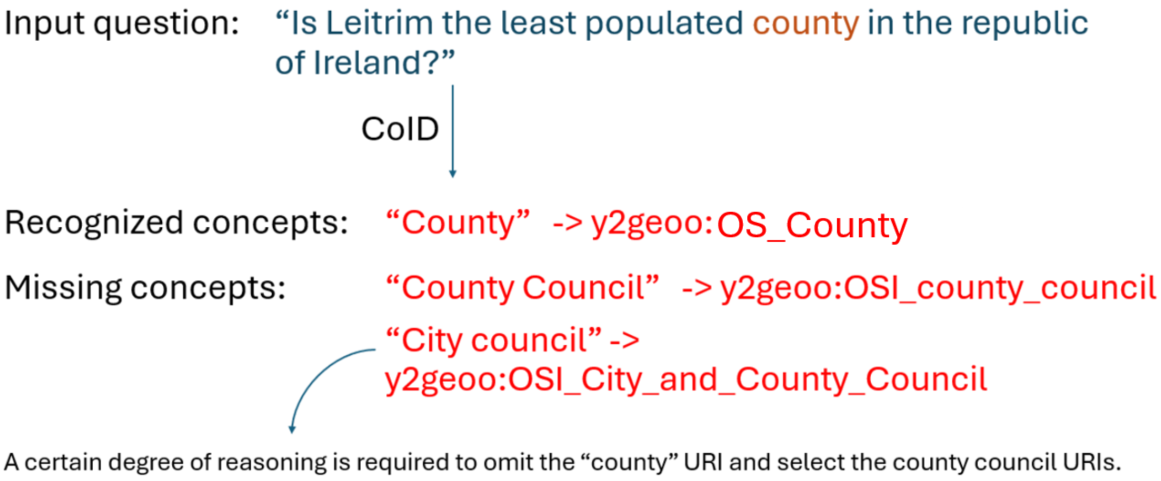
CoID

Recognized concepts: “Municipality” -> y2geoo:GAG_Municipality

“Regional Unit” -> y2geoo:GAG_RegionalUnit

Incorrect URI, part of instance and not a concept.

Figure 5.3: Incorrect concept identification due to the concept being part of an instance.



Input question: “Is Leitrim the least populated county in the republic of Ireland?”

CoID

Recognized concepts: “County” -> y2geoo:OS_County

Missing concepts: “County Council” -> y2geoo:OSI_county_council

“City council” -> y2geoo:OSI_City_and_County_Council

A certain degree of reasoning is required to omit the “county” URI and select the county council URIs.

Figure 5.4: Incorrect concept identification due to lack of ontology knowledge and reasoning.

Additionally, several other issues are evident. For instance, in Figure 5.3, two words in the sentence match labels from our list perfectly, but only one refers to an actual concept, while the other is part of a toponym. Similarly, in Figure 5.4, the word ”county” appears in the question and it is matched to a concept with the same exact label, but the required URI to answer the query has the label ”county council.” Lastly, there are cases where a URI label might be present in the input but isn’t necessary for answering the query.

These mistakes stem from the nuances of natural language. Even when a concept is critical to the question, its label might not appear directly in the input. These challenges require a level of reasoning that includes identifying whether a label is truly a concept or part of another entity, determining if additional URIs are needed, and recognizing when a URI, though not explicitly mentioned, is essential for constructing a valid query. Based on this newfound reasoning requirement the LLM-powered Concept Identifier was proposed.

LLM-powered Concept Identifiers:

- Although the LLM-CoID approach demonstrated the weakest results among the four tests, it still managed to identify nearly half of the required concepts. Interestingly, some of the desired reasoning capabilities were successfully adopted by the model. For example, if a county name appears in a question without explicitly mentioning the word ”county,” the model might still return the relevant ”county” URI concept. This occurs because the model recognizes the relevance of the concept, even when it’s not explicitly stated in the input. However, while the model successfully includes relevant URIs they are not always necessary. The model needs to posses a strong understanding of SPARQL and the underlying KG in order to successfully include only the needed URIs for the query generation phase. Clearly, this model does not have this capacity, therefore some of the related URIs that it generates end up as False Positives.

- When the model was fine-tuned on a relevant training corpus in experiment LLM-CUG, we observed a significant improvement in the true positive (TP) rate, bringing it closer to the performance of string similarity-based systems. However, this improvement was accompanied by a high rate of false positive (FP) URIs. While the model’s performance was inferior to the string similarity methods, it still showed noticeable gains over the LLM-CoID approach. This indicates the potential for LLM-power concept identifiers. Presumably, with further fine-tuning and enhanced prompt engineering we can surpass these scores.

  Furthermore, due to the small scale of the LLM, both systems struggled with hallucinations and imperfect reasoning, which contributed to their inferior performance and higher False Positive rates. Despite these shortcomings, the logical complexity of the task makes it clear that these methods represent the future of concept identification. With the utilization of a larger model, improved prompting techniques, or fine-tuning, these approaches could potentially surpass traditional string similarity-based methods.

5.6 Evaluation of SPARQL query generation with URI injection.

Following our in-depth analysis of various URI generation methods, we now turn to evaluating their impact on the performance of the SPARQL query generation system. Previous experiments demonstrated that supplying the model with the correct URIs for each question, a process known as URI-injection, significantly enhanced the system's accuracy. However, those tests were conducted under controlled conditions, relying on ground truth data to assist the model, which does not accurately represent the system's independent capabilities.

In this section, we evaluate the effectiveness of unassisted URI generation methods that we developed, which are designed to autonomously produce relevant URIs for each input question. By refining these approaches, we aim to assess whether the automatically generated URIs can genuinely enhance the performance of our SPARQL query generation system.

We will conduct three distinct experiments to evaluate the effectiveness of the generated URIs in enhancing SPARQL query generation.

1. **Instance-Only Experiment**: In the first experiment, we will focus solely on instances generated by a Named Entity Disambiguation (NED) model. Specifically, we will utilize the hybrid LLM-WAT-ED method, which delivered the most promising results in our previous tests. This experiment will help us assess how well the system performs when relying exclusively on instance URIs provided by the LLM-WAT-ED approach.
2. **Concept-Only Experiment:** Similarly to the instance-only test we will assess the importance of concept URIs for the model during SPARQL query generation. This is done by the inclusion of concept URIs identified by the GeoQA2 concept identifier (CoID), which proved to be have the best balance between TP and FP scores.
3. **Instance and Concept Experiment**: The final experiment will build upon the previous two by incorporating concepts alongside the instances. This combined approach will allow us to evaluate whether the inclusion of both types of URIs can further enhance the accuracy and effectiveness of the SPARQL query generation system or hinder it.

For this new test, we introduce a few-shot prompt containing six examples. These examples include instances with both missing and incorrect URIs, enabling the model to better adapt to the imperfections commonly found in our automatically extracted URIs. Surprisingly, this straightforward prompt also achieves a new record for unassisted SPARQL generation with Mistral, with an accuracy of 19%. This improvement can be attributed to the inclusion of six examples in the prompt, over the three previously used. While adding more examples could potentially lead to further gains, this approach lacks scientific rigor and does not provide meaningful benefits, so we will not pursue it further.

In Table 5.3 we present the results of SPARQL query generation accuracy, augmented by three URI-injection experiments as previously outlined. For comparison, we include a test on the same prompt without the addition of URI-injection. The results indicate that the model equipped with URIs performs slightly better than the baseline model, apart from the Concept URI test which actually performed worse than before.

To further validate our findings, we will replicate the same tests using the GPT-4o mini model. The results of these experiments are summarized in Table 5.4. Both Mistral and

Method	Accuracy
Mistral v0.2 7b with few-s(6)	0.19
Mistral v0.2 7b with Instance URI-i and few-s(6)	0.22
Mistral v0.2 7b with Concept URI-i and few-s(6)	0.14
Mistral v0.2 7b with Instance + Concept URI-i and few-s(6)	0.20

Table 5.3: SPARQL generation accuracy with the current best performing method and automatic URI injection methods.

GPT-4o mini appear to benefit from the inclusion of our URI-injection system. Notably, all but one of our tests demonstrated improvements over the baseline method. However, the models exhibit differing behaviors, which can be attributed to their varying capabilities. In this section, we will analyze these results in greater detail.

Mistral significantly improved its performance when only Instances were provided, with a slight improvement noted when both Instances and Concepts were supplied. Interestingly, when only Concepts were given, there was a substantial 5% drop in performance. This decline can be attributed to two key issues:

- **Imperfect prompt:** It's likely that additional prompt attempts could bring the results closer to the baseline. The examples in the current prompt may not sufficiently emphasize that the supplied concepts are not necessarily correct.
- **Low concept confidence:** Concept URIs tend to be more cryptic than Instances, leading to uncertainty in selecting the correct URI and increasing the risk of False Positive URI selection. For example, if the question is ”Is Athens north of Thessaloniki?” and the model is given the URIs ”Athens, London,” it can easily discard the False Positive. However, if supplied with URIs like ”townland, ward, district, county” for a different question, it would struggle to determine whether a necessary URI is missing or if a False Positive is present.

GPT-4o mini did not show any improvement when only Instances were supplied. However, a slight improvement was observed when concepts were provided, and most notably, the model's accuracy increased by 5%, from 31% to 36%, when both instances and concepts were introduced together.

These results may initially appear contradictory. However, when considering GPT's superior reasoning capabilities, the lack of improvement in the Instance-only experiment is likely due to the model's ability to recognize common patterns in toponym URIs, allowing it to infer them independently.

Moreover, the improvement observed when concepts are supplied supports the notion that Mistral struggles to distinguish between valid and invalid concept URIs, whereas a more advanced model like GPT can make this distinction effectively. Ultimately, both models show notable improvements when supplied with a combination of Instance and Concept URIs.

In general, these results confirm that URI-injection, contributes positively to the overall goal of SPARQL query generation. Given that the assisted URI-injection previously led to even greater performance gains, we can expect even better performance as our URI extracting modules improve.

Method	Accuracy
GPT 4o-mini with few-s(6)	0.31
GPT 4o-mini with Instance URI-i and few-s(6)	0.31
GPT 4o-mini with Concept URI-i and few-s(6)	0.32
GPT 4o-mini with Instance + Concept URI-i and few-s(6)	0.36

Table 5.4: SPARQL generation accuracy with GPT 4o-mini and URI-injection methods.

5.7 Conclusion

These results demonstrate the potential of the URI-injection method. Our efforts have initiated a hybrid system that combines GeoQA2's rule-based modules with the advanced reasoning capabilities of LLMs. Additionally, we have successfully developed an LLM-based instance identifier that surpasses the existing system in GeoQA2 by significantly reducing false positive rates. Moreover, we proposed and developed two LLM-based concept identifiers which under-perform existing string similarity methods but exhibit theoretical promise. Finally, our experiments confirmed that assisted URI injection substantially enhances the system's SPARQL query generation capabilities. This finding suggests that with further refinements to the URI-generation components, we can expect even greater improvements in the overall performance of SPARQL query generation, moving us closer to a more robust and accurate system.

6. FINE-TUNING THE SPARQL QUERY GENERATION MODEL

Fine-tuning is a critical step in machine learning, particularly in the context of transfer learning, where a pre-trained model is adapted to a specific task. Unlike training a model from scratch, which requires vast amounts of data and computational resources, fine-tuning leverages the knowledge already embedded in a pre-trained model. This process involves adjusting the model's weights slightly by training it on a smaller, task-specific dataset. By doing so, the model can specialize in the new task while retaining the generalized knowledge it gained during its initial training.

In the context of large language models (LLMs) like, our best-performing model, Mistral, fine-tuning allows the model to be adapted to domain-specific tasks, such as generating GeoSPARQL queries. The pre-trained model has already learned a wide range of linguistic patterns, knowledge, and reasoning abilities. Fine-tuning hones these capabilities to better suit the nuances and requirements of the specific domain.

6.1 Data Preparation

To maintain uniformity in our test results, we will use the same test set that was utilized in the previous accuracy measurements. This test set comprises approximately 10% of the entire dataset. The remaining 90% will be divided into an 80% training set and a 10% validation set, adhering to the 80-10-10 split commonly used for optimal model training.

6.2 Parameter-Efficient Fine-Tuning (PEFT) Techniques

Parameter-Efficient Fine-Tuning (PEFT) techniques are a set of methods designed to adapt large pre-trained models to specific tasks while minimizing the number of trainable parameters. The main idea behind PEFT is to reduce the computational and memory requirements of fine-tuning, making it feasible to adapt large models even in resource-constrained environments. PEFT techniques focus on fine-tuning only a small subset of the model's parameters, which can lead to faster training and lower risk of overfitting, especially when dealing with small datasets.

PEFT can be broken down to these fundamental categories [47]:

- **Additive Fine-Tuning:** This approach of fine-tuning involves introducing new extra trainable parameters for task-specific fine-tuning. Some methods that belong to this category include incorporating trainable adapter modules into the transformer, allowing for fine-tuning without modifying the pretrained parameters. Other techniques include soft prompts or prefix vectors appended to the input embeddings or hidden states during fine-tuning.
- **Partial Fine-Tuning:** These methods aim to reduce the number of fine-tuned parameters by selecting a subset of pre-trained parameters that are critical to downstream tasks while discarding unimportant ones.
- **Reparameterized Fine-Tuning:** Reparameterized fine-tuning methods utilize low-rank transformation to reduce the number of trainable parameters while allowing op-

erating with high-dimensional matrices (e.g., pretrained weights). These methods include LoRA, a low-rank adaptation method designed to adjust large pre-trained models to specific tasks by reducing the number of parameters that need to be fine-tuned. The main idea behind LoRA is to introduce trainable, low-rank matrices into the model's architecture, allowing for efficient task-specific adaptation without modifying the majority of the model's parameters.

A neural network contains many dense layers which perform matrix multiplication. The weight matrices in these layers typically have full rank. When adapting to a specific task, [1] shows that the pre-trained language models have a low "intrinsic dimension" and can still learn efficiently despite a random projection to a smaller subspace. Inspired by this, [16] hypothesized and proved that the updates to the weights also have a low "intrinsic rank" during adaptation. For a pre-trained weight matrix $W_0 \in \mathbb{R}^{d \times k}$, they constrain its update by representing the latter with a low-rank decomposition $W_0 + \Delta W = W_0 + BA$, where $B \in \mathbb{R}^{d \times r}$, $A \in \mathbb{R}^{r \times k}$, and the rank $r \ll \min(d, k)$. During training, W_0 is frozen and does not receive gradient updates, while A and B contain trainable parameters. Note both W_0 and $\Delta W = BA$ are multiplied with the same input, and their respective output vectors are summed coordinate-wise. This is the foundation of the LoRA fine-tuning system. Apart from providing very fast training due to the small amount of trainable parameters, this method has no inference latency, unlike many adapter methods, and can be combined with methods such as prefix-tuning.

- **Hybrid Fine-Tuning:** This approach aims to combine various PEFT methods, such as adapter, prefix-tuning and LoRA, to leverage the strengths of each method and mitigate their weaknesses. By integrating different features of PEFT methods, hybrid fine-tuning achieves improved overall performance compared to individual PEFT approaches.

- **Unified Fine-Tuning:** This category of fine-tuning methods presents a unified framework, which streamlines the incorporation of diverse fine-tuning methods into a cohesive architecture, ensuring consistency and efficiency across the adaptation and optimization of models. Unlike hybrid fine-tuning methods, this approach typically utilizes a single PEFT rather than a combination of multiple methods. As an example let us consider SparseAdapter. This method utilizes network pruning techniques to construct a unified framework in which various PEFT methods, including the adapters family and LoRA, can be further pruned to improve parameter efficiency. Specifically, SparseAdapter sets a target sparsity, denoted as s, and assigns a score, denoted as z, to all parameters of adapters and LoRA. Parameters with scores bellow the threshold z_S (corresponding to the s-th lowest percentile of z) are considered redundant and removed.

6.3 Experiments

We conducted several experiments using finetuned weights, all of which employed the QLoRA [7] approach for efficient and effective fine-tuning. QLoRA is a variant of LoRA specifically designed for quantized LLMs, enabling fine-tuning of large language models while reducing memory and computational requirements through quantization.For these tests, we fine-tuned the Mistral v0.2 7b model, as it has consistently demonstrated strong performance for this task. Due to hardware constraints, we quantized the model to 4-bits, allowing us to train the model within our available 16 GBs of VRAM.

We fine-tuned the model on the training set and evaluated its performance on the validation set, as outlined in subsection 6.1. To ensure consistency and comparability in our results, we measured the performance of all fine-tuned weights on the same test set that was used in our previous experiments. Some of our experiments were also tested on a subset of the test set which does not include particular ontologies.

We utilized supervised fine-tuning (SFT) to retrain our model. SFT relies on labeled data, making it an ideal approach for our task, which involves question/query pairs. During this process, the model learns to associate specific inputs with the correct outputs based on the examples in the training set. This method helps the model adapt to more specialized tasks, reinforcing patterns from the labeled dataset, and improving its overall performance for query generation.

Here is the experiment list along with the motivation behind each design choice:

- **Plain-4:** This experiment was designed to evaluate the model's baseline performance when trained on the entire training corpus for 4 epochs without applying any pre-processing to the training data. The goal was to understand how well the model could generalize from the raw data without any intervention or optimization.

- **Norm-4:** This experiment, like Plain-4, trains the model for 4 epochs on the entire dataset. However, in this case, we applied pre-processing to normalize the variable names within the queries. Each variable was renamed with numerical order (e.g., var0, var1, var2) to prevent the model from being penalized for incorrectly predicting variable names, which are not crucial for generating valid SPARQL queries. This approach aimed to improve the model's ability to focus on the structural and semantic aspects of query generation, rather than irrelevant details like variable naming.

- **Norm-Simple-5:** This experiment follows the same procedure as Norm-4 but extends the training to 5 epochs. Additionally, the ”Simple” designation indicates that questions that originate from YAGO2 were excluded from the training set. These questions typically use the rdfs vocabulary in their queries. RDFS or Resource Description Framework Schema is a set of classes with certain properties using the RDF extensible knowledge representation data model, providing basic elements for the description of ontologies. The rationale behind this decision was to allow the model to better focus on the more straightforward YAGO2geo SPARQL queries, which form the bulk of the dataset. Queries with the rdfs vocabulary were more complex and relatively rare, which likely led to confusion during training. Experiments tagged with ”Simple” will be evaluated both on the full test set and on a subset without any rdfs vocabulary queries.

- **Norm-Simple-Trunc-5:** In this experiment, additional pre-processing is applied to the training corpus. The training set contains queries with mixed representations of URIs, some using the full URI, while others use the prefix form. This inconsistency can confuse the model, which does not need to learn the full URI when the prefix form suffices. Moreover, penalizing the model for using the ”wrong” URI form (even though both forms result in valid SPARQL queries) can negatively impact performance. To resolve this, we truncated all full URI links to their prefix form (e.g., `"http://yago-knowledge.org/resource/Sfakia"` becomes `"yago:Sfakia"`).

Method	Test Set Accuracy	Simple Set Accuracy
Plain-4	0.13	-
Norm-4	0.20	-
Norm-Simple-5	0.19	0.21
Norm-Simple-Trunc-5	0.31	0.33
Norm-Simple-Trunc-8	0.39	0.43

Table 6.1: Resulting accuracy of fine-tuned Mistral models on the full test set and a simpler subset lacking rdfs ontologies.

- **Norm-Simple-Trunc-8:** This final experiment is the same as Norm-Simple-Trunc-5 with an extended training of 8 epochs.

6.4 Results

The results of the previously analyzed experiments are documented in Table 6.1. In the ”Test Set Accuracy” column, we present the accuracy of each model on the full test set, which is a randomly selected subset of GeoQuestions1089 and reflects the models’ overall capabilities on the dataset. The ”Simple Set Accuracy” column lists the accuracy of the ’Simple’ designated models on a specific subset of the test set that excludes queries containing the rdfs ontology.

Initially, we observed a 13% accuracy from the Plain-4 method. This notable result demonstrates that fine-tuning is effective, as the same pre-trained model performed with 0% accuracy when evaluated with the same zero-shot prompt. This improvement confirms that the model has adapted to the task through fine-tuning.

The Norm-4 method further refines this approach. Our intuition that the variable names are unnecessary during training is validated by a significant 7% increase in accuracy, raising the system’s performance to 20%. This improvement demonstrates the effectiveness of normalizing variable names, as it helps the model focus on more relevant aspects of the SPARQL query generation task.

The Norm-Simple-5 method did not exhibit any improvement over Norm-4. While it seems plausible that the presence of a few complex ontology examples in the training set could confuse the model, omitting these examples did not result in performance gains. This suggests that the model may not have been heavily impacted by the complexity of these queries, or that removing them did not provide the anticipated clarity in understanding the simpler YAGO2geo SPARQL queries.

The introduction of truncation in the training set led to significant performance improvements in the Norm-Simple-Trunc-5 experiment. The accuracy increased to 31% on the entire test set and 33% on YAGO2geo-specific questions. This suggests that simplifying the URI format helped the model focus on more critical aspects of the task. Furthermore, extending the training to 8 epochs in the Norm-Simple-Trunc-8 experiment further enhanced the model’s capabilities, achieving 39% accuracy on the entire test set and 43% on the simple set, demonstrating the benefits of prolonged fine-tuning and consistent preprocessing. It is important to note that we limit ourselves to 8 epochs of training in this work due to computational and time constraints.

These results are extremely promising as they even surpass the performance of GPT-4o mini with the URI-injection method which results in only 36% accuracy.

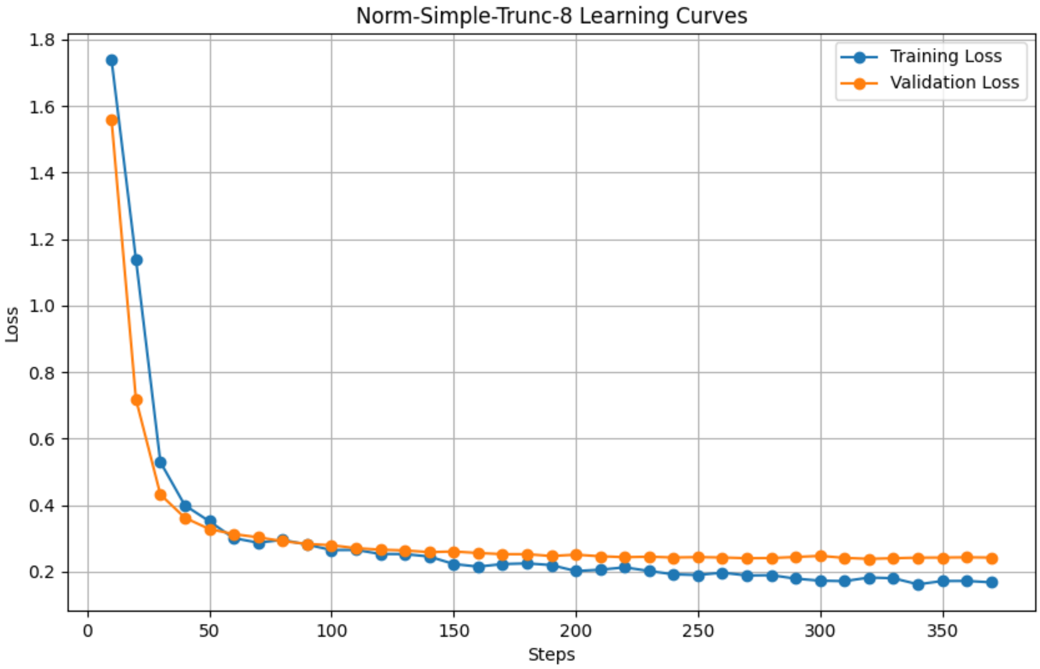


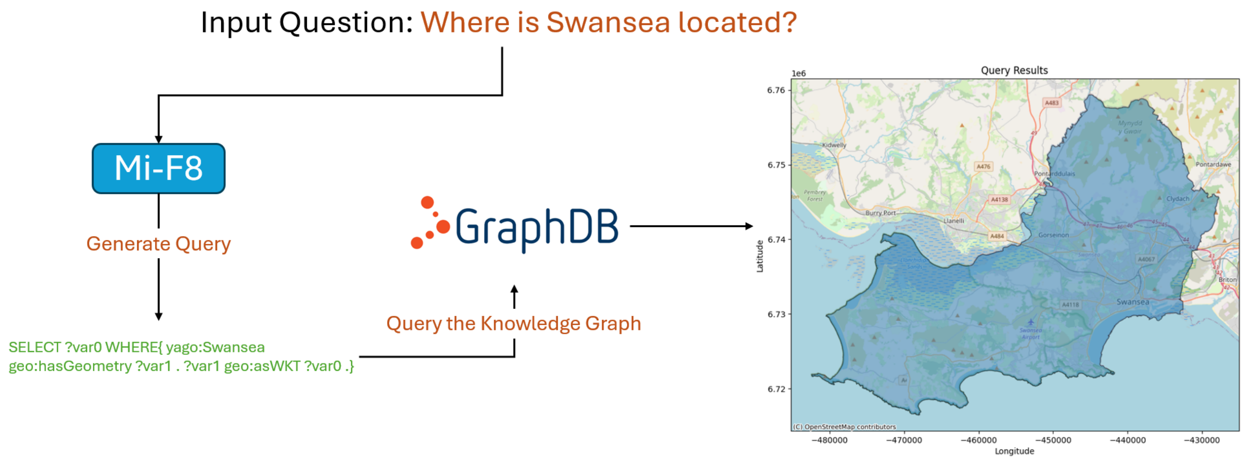
Figure 6.1: Learning Curves for the Norm-Simple-Trunc-8 model.

In Figure 6.1, we observe the learning curves of the Norm-Simple-Trunc-8 model. Both training and validation losses decrease rapidly during the initial steps and then gradually stabilize, indicating that the model is converging well. The small and consistent gap between the losses suggests that overfitting has not occurred. Interestingly, while the validation loss between 5 epochs (250 steps) and 8 epochs appears almost unchanged, the accuracy improvement with 8 epochs is substantial. This suggests that the loss values might not be the most reliable metric for evaluating the system's performance, and continued training could further refine the model's weights for even better outcomes.

We also attempted to apply the same six-example few-shot approach that previously yielded promising results, along with URI-injection, on our fine-tuned model. However, these attempts resulted in lower accuracies than the zero-shot question prompts and, thus, are not documented. This outcome was anticipated, as the model no longer needs to rely on in-context learning for SPARQL generation; it was trained on a significantly larger corpus. In fact, the more extensive prompts, which included these few-shot examples, increased the complexity of the questions and did not align with the format the model was trained on, ultimately leading to confusion and a decrease in accuracy.

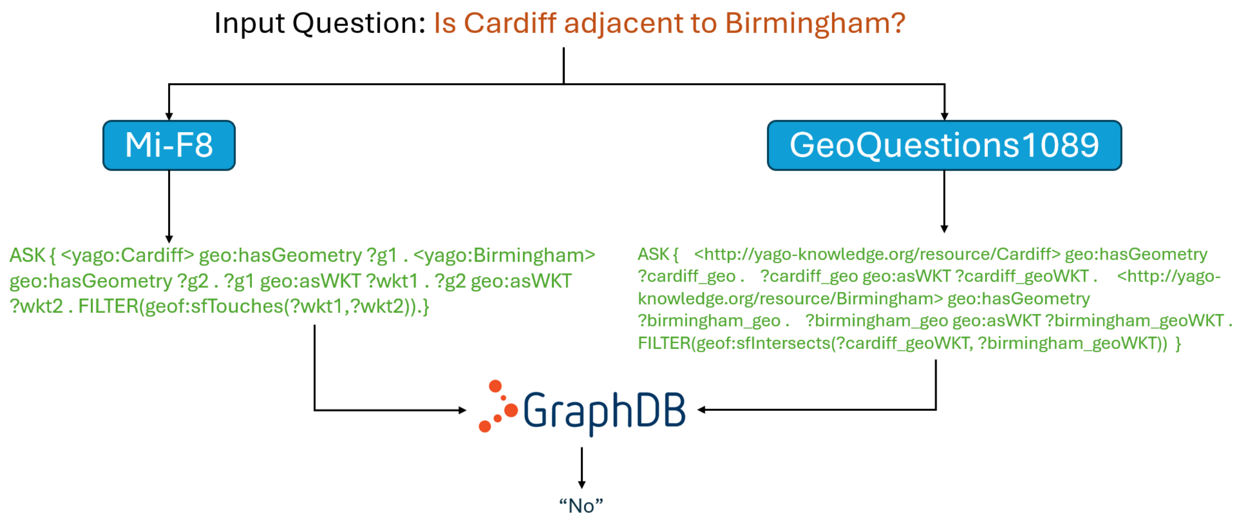
Another idea was to test the model with URI-injection without the extensive few-shot examples. This approach would keep the prompt's complexity similar to the zero-shot settings and maintain a format close to what the model was trained on. However, performance still dropped, likely because the absence of examples led the model to overlook the possibility that the injected URIs could be false or missing. In previous experiments, this issue was mitigated by including multiple examples that highlighted these characteristics.

Another possible explanation is that the fine-tuned model has encountered many URIs and can independently infer them better than our URI-injection methods. This suggests the potential for a more powerful URI-injection system based on a fine-tuned LLM. In summary, while the URI-injection method has proven effective as a prompt engineering strategy for pre-trained models, fine-tuned models are specifically calibrated for a task, making them



```
SELECT ?var0 WHERE{ yago:Swansea
geo:hasGeometry ?var1 . ?var1 geo:asWKT ?var0 .}
```

Figure 6.2: Example of the question-answering capabilities of the Mi-F8 model.



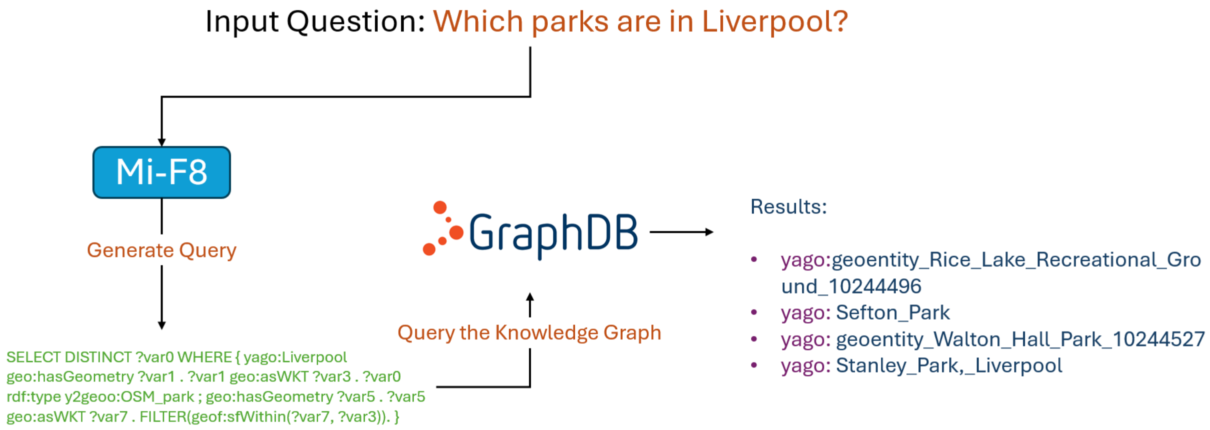
```
ASK { <yago:Cardiff> geo:hasGeometry ?g1 . <yago:Birmingham>
geo:hasGeometry ?g2 . ?g1 geo:asWKT ?wkt1 . ?g2 geo:asWKT
?wkt2 . FILTER(geof:sfTouches(?wkt1,?wkt2)).}
```

```
ASK {   <http://yago-knowledge.org/resource/Cardiff> geo:hasGeometry
?cardiff_geo .   ?cardiff_geo geo:asWKT ?cardiff_geoWKT .   <http://yago-
knowledge.org/resource/Birmingham> geo:hasGeometry
?birmingham_geo .   ?birmingham_geo geo:asWKT ?birmingham_geoWKT .
FILTER(geof:sfIntersects(?cardiff_geoWKT, ?birmingham_geoWKT)) }
```

Figure 6.3: Comparison of generated and ground-truth ASK query for a given question.

less responsive to fluctuations in the prompt that do not enhance their efficiency.

In Figures 6.2, 6.3, 6.4 we present various examples of questions successfully answered by our best finetuned model, Norm-Simple-Trunc-8, referred to as **”Mi-F8”**, along with their respective results extracted from the Knowledge Graph. In Figure 6.3 we note the difference between the generated and the ground-truth ASK query. While both are valid, there are some minor differences in their structure and the variable naming conventions, which is to be expected from normalization applied to the variable names during finetuning.


Input Question: Which parks are in Liverpool?
Mi-F8
Generate Query
GraphDB
Query the Knowledge Graph
Results:

```
SELECT DISTINCT ?var0 WHERE { yago:Liverpool
geo:hasGeometry ?var1 . ?var1 geo:asWKT ?var3 . ?var0
rdf:type y2geoo:OSM_park ; geo:hasGeometry ?var5 . ?var5
geo:asWKT ?var7 . FILTER(geof:sfWithin(?var7, ?var3)). }
```

- yago:geoentity_Rice_Lake_Recreational_Ground_10244496
- yago: Sefton_Park
- yago: geoentity_Walton_Hall_Park_10244527
- yago: Stanley_Park,_Liverpool

Figure 6.4: Example of the question-answering capabilities of the Mi-F8 model.

7. CONCLUSIONS AND FUTURE WORK

We present how our systems compare with GeoQA2. Furthermore, we will provide a concise summary focusing on the key outcomes of this work, while also highlighting potential avenues for future research and development.

7.1 EVALUATION OF OUR SYSTEMS

In Table 7.1 we present a comparison between GeoQA2 and two of our approaches using the GeoQuestions1089 dataset. It's important to clarify that GeoQA2 was evaluated on the complete dataset, whereas our methods were assessed only on a randomly sampled subset due to training constraints. However, since this test set consists of 10% of the entire dataset, randomly selected, the accuracy results for our system should closely reflect its overall performance. Specifically, we will be comparing GeoQA2 with our URI-injection prompting approach used with GPT-4o mini which includes both concept and instance URIs. This approach will be denoted as ”**Ui-GPT**”. The second approach that we compare is our best performing fine-tuned Mistral v0.2 7b model which was trained for 8 epochs on our carefully pre-processed training set. This model will be denoted as ”**Mi-F8**”.

We observe that Ui-GPT achieves performance levels comparable to GeoQA2, making it an excellent option for easy implementation. As LLMs continue to advance and improve their reasoning abilities and understanding of SPARQL, this method could deliver even better results, especially with a more sophisticated URI-generation system, all without the need for costly fine-tuning. This adaptability allows its performance to be applied across various tasks and ontologies.

More significantly, our Mi-F8 model matches GeoQA2's performance while being a compact 7 billion parameter model, quantized to 4-bit precision. This low resource requirement means it is not hardware or computation-intensive, making it a viable replacement for GeoQA2. Additionally, with further training or a larger dataset, this model has the potential to achieve even more impressive results.

7.2 SUMMARY AND FUTURE WORK

The key outcomes of this work are:

- A thorough evaluation and comparison of various LLMs on the task of GeoSPARQL query generation.
- The development of the URI-injection prompt engineering method, which offers significant performance improvements on pre-trained models, proving to be powerful and require no expensive training.

Method	Accuracy
GeoQA2	38.54%
Ui-GPT	36%
Mi-F8	39%

Table 7.1: Comparison of accuracy scores between GeoQA2 and our methods.

- The creation of Mi-F8, a fine-tuned LLM based on Mistral, that matches state-of-the-art performance in GeoSPARQL query generation on the GeoQuestions1089 dataset. The model is quantized to 4-bit precision, making it lightweight for fast inference on standard computers.

Some promising directions for future improvements of this work are:

- The creation of a unified benchmark for accurate GeoSPARQL query similarity, which could better capture nuances missed by the BLEU similarity metric, especially in cases where evaluating based on query results is not feasible.
- Developing more advanced URI-generation systems for URI-injection could enhance performance. Fine-tuned LLMs capable of generating all necessary URIs for a query, not limited to instances or concepts, could outperform current approaches.
- Fine-tuning larger versions of the Mistral v0.2 model on the same dataset may lead to improved performance. Further training of the Mi-F8 model could also yield better results, as no overfitting was observed, though training was limited to 8 epochs due to computational constraints.
- Mi-F8 was fine-tuned with zero-shot examples, and as a result, it performs best when given zero-shot examples during inference. In fact, when provided with few-shot examples, its performance declines, as it is not optimized for that format. A potential future direction to improve the system would be to fine-tune the model with differently formatted training corpora, such as zero-shot chain-of-thought or few-shot prompting. This would allow it to handle those formats during inference, potentially leading to better results.

As LLMs continue to enhance their understanding of SPARQL, improve in reasoning abilities, and become more efficient, they are poised to revolutionize the functioning of Question Answering systems, delivering unprecedented results. Additionally, as GeoSPARQL datasets based on YAGO2geo evolve, these advancements will further amplify the impact of these systems, enabling even more accurate and insightful answers.

ABBREVIATIONS - ACRONYMS

LLM	Large Language Model
NLP	Natural Language Processing
SPARQL	SPARQL Protocol and RDF Query Language
GeoSPARQL	Extension of SPARQL that includes geospatial relations
YAGO	Yet Another Great Ontology - Database
CoT	Chain-of-Thought (prompting technique)
KG	Knowledge Graph
URI	Uniform Resource Identifier
PEFT	Parameter-Efficient Fine-Tuning
LoRA	Low Rank Adaptation
QLoRA	Quantized Low-Rank Adaptation
GeoQA2	Geospatial Question Answering system by NKUA
SOTA	State-of-the-art
POS	Part-of-speech tagging
WAT	Entity linker tool
NED	Named-Entity Disambiguation
NER	Named-Entity Recognition
TP/FP	True Positives/False Positives
W3C	World Wide Web Consortium
NMT	Neural Machine Translation
LSTM	Long short-term memory - type of recurrent neural network
it	Instruction-tuned (in the context of specific LLMS)
RDF	Resource Description Framework
POI	Points of Interest
OSGen	Overall SPARQL Generation
InUK	Inherent URI Knowledge
BLEU	Bilingual Evaluation Understudy
0-s	Zero-shot learning(prompting technique)
URI-i	URI-injection (prompting technique)

Geo-i	GeoSPARQL ontology-injection (prompting technique)
few-s	Few-shot learning (prompting technique)
P-URI	Naive Popular-URI selection
LLM-ED	LLM-powered disambiguation for URI selection
LLM-IUG	Large Language Model - Instance URI Generation
IE-WAT	Instance Identifier with WAT and DPT
WAT-II	Instance Identifier with WAT
DPT	Dependency Parse Tree
REL-II	Instance Identifier with REL
LLM-WAT-ED	LLM-powered entity recognition with WAT entity linking
CoID	Concept Identifier
Plain-CoID	Plain-Concept Identifier
LLM-CoID	LLM-powered Concept Identifier
LLM-CUG	Large Language Model-Concept URI Generation
Plain-4	no preprocessing and 4 epochs training
Norm-4	normalized variable names and 4 epochs training
Norm-Simple-5	Normalized variable names, limited training examples and 5 epochs training
Norm-Simple-Trunc-5/8	Normalized variable names, limited training examples and truncated URIs for 5 or 8 epochs respectively
Ui-GPT	URI-injection used with GPT-4o mini
Mi-F8	Mistral-Finetuned for 8 epochs (Norm-Simple-Trunc-8)

APPENDIX A. PROMPTS USED IN THIS WORK

A.1 PROMPTS USED DURING EMPIRICAL EVALUATION

Zero-shot prompts with simple instructions:

- ”Generator is an expert SPARQL query generator. For each question that the user supplies, the generator will convert it into a valid SPARQL query that can be used to answer the question.
  Human: ”Where is the Dorset county located?”
  Generator:”
- ”Generator creates valid SPARQL queries.The user will provide a question and the generator will convert it into an equivalent SPARQL query that answers the user’s question.
  Human: ”What is the population of Aegina ?”
  Generator:”

Same simple zero-shot prompt but we specify that the query should target the YAGO KG:

- ”Generator is an expert SPARQL query generator. For each question that the user supplies, the generator will convert it into a valid SPARQL query that can be used to answer the question. The query will be based on the YAGO knowledge graph.
  Human: ”Where is the Dorset county located?”
  Generator:”

One-shot prompt with the same simple instructions for SPARQL query generation:

- ”Generator is an expert SPARQL query generator. For each question that the user supplies, the generator will convert it into a valid SPARQL query that can be used to answer the question.
  Human: ”Where is Oxfordshire located?”
  Generator: ”SELECT ?WKT WHERE yago:Oxfordshire geo:hasGeometry ?o. ?o geo:asWKT ?WKT. ”
  Human: ”What is the total area of County Galway?”
  Generator:”
- ”Generator is an expert SPARQL query generator. For each question that the user supplies, the generator will convert it into a valid SPARQL query that can be used to answer the question. The query will be based on the YAGO knowledge graph.
  Human: ”What is the total area of Glengarra Wood forest?”
  Generator: ”select distinct (strdf:area(?geoWKT) as ?area) where <http://yago-knowledge.org/resource/geoentity _Glengarra_Wood_3300941> geo:hasGeometry ?o. ?o geo:asWKT ?geoWKT. ”
  Human: ”What is the population of Piraeus?”
  Generator:”

Same approach with three-shot prompting:

- ”Generator is an expert SPARQL query generator. For each question that the user supplies, the generator will convert it into a valid SPARQL query that can be used to answer the question.
  Human: ”Where is Oxfordshire located?”
  Generator: ”SELECT ?WKT WHERE yago:Oxfordshire geo:hasGeometry ?o. ?o geo:asWKT ?WKT. ”
  Human: ”What is Dublin’s administrative type?”
  Generator: ”select ?e where yago:Dublin rdf:type ?e ”
  Human: ”What population does Icaria have?”
  Generator: ”SELECT ?population WHERE yago:Icaria y2geoo:hasGAG_Population ?population . ”
  Human: ”Where is Scotland located?”
  Generator:”

More complex examples with three-shot prompting:

- ”Generator is an expert SPARQL query generator. For each question that the user supplies, the generator will convert it into a valid SPARQL query that can be used to answer the question.
  Human: ”Which localities are located east of forests in County Wicklow?”
  Generator: ”SELECT DISTINCT ?a WHERE yago:County_Wicklow geo:hasGeometry ?o . ?o geo:asWKT ?geoWKT . ?a rdf:type y2geoo:OSM_locality; geo:hasGeometry ?o1 . ?o1 geo:asWKT ?geoWKT1 . ?b rdf:type y2geoo:OSM_forest; geo:hasGeometry ?o2 . ?o2 geo:asWKT ?geoWKT2 . FILTER (strdf:within(?geoWKT1, ?geoWKT) && strdf:within(?geoWKT2, ?geoWKT) && strdf:right(?geoWKT1, ?geoWKT2)) ”
  Human: ”Is there a stream located east of a lake in Corfu?”
  Generator: ”ASK yago:Corfu geo:hasGeometry ?o2 . ?o2 geo:asWKT ?xWKT2 . ?x2 rdf:type y2geoo:OSM_stream . ?x2 geo:hasGeometry ?x2Geom. ?x2Geom geo:asWKT ?iWKT2. ?x1 rdf:type y2geoo:OSM_lake . ?x1 geo: hasGeometry ?x1Geom. ?x1Geom geo:asWKT ?iWKT1. FILTER(geof: sfWithin (?iWKT1, ?xWKT2) && geof: sfWithin(?iWKT2,?xWKT2) && strdf:right(?iWKT2,?iWKT1)) ”
  Human: ”Which region of Greece has the most inhabitants?”
  Generator: ”SELECT DISTINCT ?region WHERE ?region rdf:type y2geoo:GAG_Region . ?region y2geoo:hasGAG_Population ?population ORDER BY DESC (?population) LIMIT 1”
  Human: ”Which county in the British Isles is the smallest by area?”
  Generator:”

Same complex three-shot prompting approach with explicit instructions to target the YAGO KG:

- ”Generator is an expert SPARQL query generator. For each question that the user supplies, the generator will convert it into a valid SPARQL query that can be used to answer the question. The query will be based on the YAGO knowledge graph.
  Human: ”Which are the 2 newest bridges of Ireland?”
  Generator: ”SELECT DISTINCT ?bridge WHERE ?type rdfs:subClassOf+ yago:wordnet _bridge _102898711. ?bridge a ?type .?bridge yago:isLocatedIn+ yago:California.?bridge yago: wasCreatedOnDate ?date. ORDER BY DESC(?date) LIMIT 2”
  Human: ”Is there a stream located east of a lake in Corfu?”
  Generator: ”ASK yago:Corfu geo:hasGeometry ?o2 . ?o2 geo:asWKT ?xWKT2

. ?x2 rdf:type y2geoo:OSM_stream . ?x2 geo:hasGeometry ?x2Geom. ?x2Geom geo:asWKT ?iWKT2. ?x1 rdf:type y2geoo:OSM_lake . ?x1 geo:hasGeometry ?x1Geom. ?x1Geom geo:asWKT ?iWKT1. FILTER(geof:sfWithin(?iWKT1, ?xWKT2) && geof: sfWithin (?iWKT2,?xWKT2) && strdf:right(?iWKT2,?iWKT1)) ”
Human: ”Which region of Greece has the most inhabitants?”
Generator: ”SELECT DISTINCT ?region WHERE ?region rdf:type y2geoo:GAG_Region . ?region y2geoo:hasGAG_Population ?population ORDER BY DESC (?population) LIMIT 1”
Human: ”Is Leitrim the least populated county in the Republic of Ireland?”
Generator:”

A.2 PROMPTS USED DURING STRING SIMILARITY AND RESULTS-BASED EVALUATION

Zero-shot prompt (0-s):

- ”Generator is an expert SPARQL query generator. For each question that the user supplies, the generator will convert it into a valid SPARQL query that can be used to answer the question. The query will be based on the DBpedia knowledge graph. The query should be enclosed by three backticks on new lines, denoting that it is a code block.
  Human: [DATASET QUESTION]
  Generator: “”

Zero-shot prompt with URI-injection (0-s, URI-i):

- ”Generator is an expert SPARQL query generator. For each question that the user supplies, the generator will convert it into a valid SPARQL query that can be used to answer the question. The query will be based on the DBpedia knowledge graph. The query should be enclosed by three backticks on new lines, denoting that it is a code block.
  Human: [DATASET QUESTION]
  The generator must use these URIs to answer the question: [GOLD STANDARD URIS]
  Generator: “”

Zero-shot prompt with URI-injection and GeoSPARQL description(0-s, URI-i, Geo-i):

- ”Generator is an expert SPARQL query generator. For each question that the user supplies, the generator will convert it into a valid SPARQL query that can be used to answer the question. The query will be based on the DBpedia knowledge graph. The query should be enclosed by three backticks on new lines, denoting that it is a code block.
  The resulting query may have to be in GeoSPARQL. The GeoSPARQL ontology is defined by: URI: http://www.opengis.net/ont/geosparql Classes: Feature, Feature Collection, Geometry, Geometry Collection, Spatial Object, Spatial Object Collection
  Object Properties: default geometry, contains, covered by, covers, disjoint, equals,

inside, meet, overlap, has area, has bounding box, has centroid, has default geometry, has geometry, has length, has perimeter length, has size, has spatial accuracy, has spatial resolution, has volume, disconnected, externally connected, equals, non-tangential proper part, non-tangential proper part inverse, partially overlapping, tangential proper part, tangential proper part inverse, contains, crosses, disjoint, equals, intersects, overlaps, touches, within
Datatype Properties: as DGGS, as GML, as GeoJSON, as KML, as WKT, coordinate dimension, dimension, has area in square meters, has length in meters, has perimeter length in meters, has metric size, has spatial accuracy in meters, has spatial resolution in meters, has volume in cubic meters, has serialization, is empty, is simple, spatial dimension
Human: [DATASET QUESTION]
The generator must use these URIs to answer the question: [GOLD STANDARD URIS]
Generator: “‘”

Few-shot prompt (3 examples) with URI-injection (few-s(3), URI-i):

- ”Generator is an expert SPARQL query generator. For each question that the user supplies, the generator will convert it into a valid SPARQL query that can be used to answer the question. The query will be based on the DBpedia knowledge graph. The query should be enclosed by three backticks on new lines, denoting that it is a code block.
  Human: In Breckland district, which forests are south of streams?
  The generator must use these URIs to answer the question: [’yago:Breckland_District’, ’geo: hasGeometry’, ’geo:asWKT’, ’rdf:type’, ’y2geoo:OSM_forest’, ’geo: hasGeometry’, ’geo:asWKT’, ’rdf:type’, ’y2geoo:OSM_stream’, ’geo:hasGeometry’, ’geo:asWKT’, ’strdf:within’, ’strdf:within’, ’strdf:below’]
  Generator: “‘SELECT DISTINCT ?forest WHERE yago:Breckland _District geo: hasGeometry ?o1 . ?o1 geo:asWKT ?geoWKT1 . ?forest rdf:type y2geoo:OSM_forest . ?forest geo: hasGeometry ?o2 . ?o2 geo:asWKT ?geoWKT2 . ?stream rdf:type y2geoo: OSM_stream . ?stream geo: hasGeometry ?o3 . ?o3 geo:asWKT ?geoWKT3 . FILTER (strdf:within (?geoWKT2, ?geoWKT1) && strdf:within (?geoWKT3, ?geoWKT1) && strdf:below (?geoWKT2, ?geoWKT3)) “‘
  Human: How many streams intersect with lakes?
  The generator must use these URIs to answer the question: [’rdf:type’, ’y2geoo: OSM _stream’, ’geo:hasGeometry’, ’geo:asWKT’, ’rdf:type’, ’y2geoo:OSM_lake’, ’geo: hasGeometry’, ’geo:asWKT’, ’geof:sfIntersects’]
  Generator: “‘SELECT (COUNT (DISTINCT ?p1) as ?streams) WHERE ?p1 rdf:type y2geoo:OSM_stream; geo:hasGeometry ?p1geo. ?p1geo geo:asWKT ?p1WKT. ?p2 rdf:type y2geoo:OSM_lake; geo:hasGeometry ?p2geo. ?p2geo geo:asWKT ?p2WKT. FILTER(geof:sfIntersects(?p1WKT, ?p2WKT)) “‘
  Human: Which Municipalities are on Thessaly’s border?
  The generator must use these URIs to answer the question: [’yago: Thessaly’, ’geo: hasGeometry’, ’geo:asWKT’, ’rdf:type’, ’y2geoo: GAG_Municipality’, ’geo: hasGeometry’, ’geo: asWKT’, ’strdf:touches’] Generator: “‘SELECT distinct ?rg where yago: Thessaly geo: hasGeometry ?tgeo . ?tgeo geo:asWKT ?tgWKT . ?rg rdf:type y2geoo: GAG _Municipality . ?rg geo: hasGeometry ?rggeo . ?rggeo geo:asWKT ?rgWKT . FILTER (strdf: touches (?tgWKT,?rgWKT)) . “‘
  Human: [DATASET QUESTION]

The generator must use these URIs to answer the question: [GOLD STANDARD URIS]
Generator: “‘”

A.3 CHAIN-OF-THOUGH PROMPTS USED

CoT prompt with 3 examples:

- ”Generator is an expert SPARQL query generator. For each question that the user supplies, the generator will convert it into a valid SPARQL query that can be used to answer the question. The query should be enclosed by three backticks on new lines, denoting that it is a code block. The generator is logical and creates each query by first explaining its thought process step-by-step.
  Human: Where is Swansea located?
  Generator: Let’s think step by step. First, we want to find the location of Swansea, so we start with select ?geoWKT to specify that we need the geographic data. Next, we need to identify how Swansea’s location is stored in the knowledge graph. We use yago:Swansea geo:hasGeometry ?o. to find the geometric data related to Swansea. Then, we need to extract the specific coordinates. We do this with ?o geo:asWKT ?geoWKT. to get the Well-Known Text (WKT) representation of Swansea’s geometry. Finally, we wrap these patterns in a where clause to structure our query properly. The final result: “‘select ?geoWKT where yago:Swansea geo:hasGeometry ?o. ?o geo:asWKT ?geoWKT. “‘
  Human: Which Greek regions have between 500000 and 1000000 inhabitants?
  Generator: Let’s think step by step. First, we want to find Greek regions with a population between 500,000 and 1,000,000, so we start with select ?region to specify that we need the region names. Next, we need to identify which entities are Greek regions. We use ?region a y2geoo:GAG_Region to find entities classified as Greek regions. Then, we need to get the population of these regions. We do this with ?region y2geoo:hasGAG_Population ?pop to find the population data associated with each region. After that, we need to filter the results to only include regions with populations between 500,000 and 1,000,000. We use filter(?pop < 1000000) to exclude regions with more than 1,000,000 inhabitants and filter(?pop > 500000) to exclude regions with fewer than 500,000 inhabitants. Finally, we wrap these patterns in a where clause to structure our query properly. The final result: “‘select ?region where ?region a y2geoo: GAG_Region . ?region y2geoo: hasGAG_Population ?pop. filter(?pop < 1000000). filter(?pop > 500000). “‘
  Human: Is Doolin to the south of Dublin?
  Generator: Let’s think step by step. Question asks for yes/no answer: Use ASK query Need to compare locations: Retrieve geometric data for both geo:hasGeometry and geo:asWKT predicates for Doolin and Dublin Check if one is south of the other: Use geospatial comparison function FILTER with strdf:below function Steps to build query: a. Get Doolin’s geometry: http:// yago-knowledge.org/ resource/ Doolin b. Get Dublin’s geometry: http:// yago-knowledge.org/ resource/ Dublin c. Compare using FILTER and strdf:below Resulting query: “‘ASK <http:// yago-knowledge.org/ resource/ Doolin> geo: hasGeometry ?o. ?o geo:asWKT ?geoWKT. <http:// yago-knowledge.org/ resource/ Dublin> geo: hasGeometry ?o1. ?o1 geo:asWKT ?geoWKT1. FILTER(strdf:below(?geoWKT,?geoWKT1)) “‘
  Human: [DATASET QUESTION]

Generator: Let's think step by step."

CoT prompt with 3 examples designed for automatic URI-injection:

- "Generator is an expert SPARQL query generator. For each question that the user supplies, the generator will convert it into a valid SPARQL query that can be used to answer the question. The generator may be provided with a list of URIs. Some of these URIs are relevant, while others are not. The generator must carefully identify and use only the correct URIs when they are provided. If the correct URIs are not available, the generator will rely on its understanding to construct the appropriate query. The query should be enclosed by three backticks on new lines, denoting that it is a code block. The generator is logical and creates each query by first explaining its thought process step-by-step.

  Human: Where is Swansea located?
  Provided URIs: yago:Swansea,y2geoo:OS_UnitaryAuthority
  Generator: Let's think step by step. First, we want to find the location of Swansea, so we start with select ?geoWKT to specify that we need the geographic data. Next, we need to identify how Swansea's location is stored in the knowledge graph. We use yago:Swansea geo:hasGeometry ?o. to find the geometric data related to Swansea. Then, we need to extract the specific coordinates. We do this with ?o geo:asWKT ?geoWKT. to get the Well-Known Text (WKT) representation of Swansea's geometry. Finally, we wrap these patterns in a where clause to structure our query properly. The final result: "'select ?geoWKT where yago:Swansea geo:hasGeometry ?o. ?o geo:asWKT ?geoWKT. "'

  Human: Which Greek regions have between 500000 and 1000000 inhabitants?
  Provided URIs: yago:Greece
  Generator: Let's think step by step. First, we want to find Greek regions with a population between 500,000 and 1,000,000, so we start with select ?region to specify that we need the region names. Next, we need to identify which entities are Greek regions. We use ?region a y2geoo:GAG_Region to find entities classified as Greek regions. We do not need the provided URI of Greece. Then, we need to get the population of these regions. We do this with ?region y2geoo:hasGAG_Population ?pop to find the population data associated with each region. After that, we need to filter the results to only include regions with populations between 500,000 and 1,000,000. We use filter(?pop < 1000000) to exclude regions with more than 1,000,000 inhabitants and filter(?pop > 500000) to exclude regions with fewer than 500,000 inhabitants. Finally, we wrap these patterns in a where clause to structure our query properly. The final result: "'select ?region where ?region a y2geoo:GAG_Region . ?region y2geoo:hasGAG_Population ?pop. filter(?pop < 1000000). filter(?pop > 500000). "'

  Human: Is Doolin to the south of Dublin?
  Provided URIs: yago:Doolin,yago:Dublin
  Generator: Let's think step by step. Question asks for yes/no answer: Use ASK query Need to compare locations: Retrieve geometric data for both geo:hasGeometry and geo:asWKT predicates for Doolin and Dublin Check if one is south of the other: Use geospatial comparison function FILTER with strdf:below function Steps to build

query: a. Get Doolin’s and Dublin’s geometry from the provided URIs: yago:Doolin and yago:Dublin b. Compare using FILTER and strdf:below The final result: “‘ASK <http://yago-knowledge.org/resource/Doolin> geo:hasGeometry ?o. ?o geo:asWKT ?geoWKT. <http://yago-knowledge.org/resource/Dublin> geo:hasGeometry ?o1. ?o1 geo:asWKT ?geoWKT1. FILTER(strdf:below(?geoWKT,?geoWKT1)) “‘

Human: [DATASET QUESTION]
Provided URIs: [GENERATED URIS]
Generator: Let’s think step by step.”

BIBLIOGRAPHY

[1] Zeyuan Allen-Zhu, Yuanzhi Li, and Zhao Song. A convergence theory for deep learning via over-parameterization, 2019.

[2] Ebtesam Almazrouei, Hamza Alobeidli, Abdulaziz Alshamsi, Alessandro Cappelli, Ruxandra Cojocaru, Mérouane Debbah, Étienne Goffinet, Daniel Hesslow, Julien Launay, Quentin Malartic, Daniele Mazzotta, Badreddine Noune, Baptiste Pannier, and Guilherme Penedo. The falcon series of open language models. *CoRR*, abs/2311.16867, 2023.

[3] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary G. Ives. Dbpedia: A nucleus for a web of open data. In Karl Aberer, Key-Sun Choi, Natasha Fridman Noy, Dean Allemang, Kyung-Il Lee, Lyndon J. B. Nixon, Jennifer Golbeck, Peter Mika, Diana Maynard, Riichiro Mizoguchi, Guus Schreiber, and Philippe Cudré-Mauroux, editors, *The Semantic Web, 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007, Busan, Korea, November 11-15, 2007*, volume 4825 of *Lecture Notes in Computer Science*, pages 722–735. Springer, 2007.

[4] Debayan Banerjee, Pranav Ajit Nair, Jivat Neet Kaur, Ricardo Usbeck, and Chris Biemann. Modern baselines for SPARQL semantic parsing. In Enrique Amigó, Pablo Castells, Julio Gonzalo, Ben Carterette, J. Shane Culpepper, and Gabriella Kazai, editors, *SIGIR '22: The 45th International ACM SIGIR Conference on Research and Development in Information Retrieval, Madrid, Spain, July 11 - 15, 2022*, pages 2260–2265. ACM, 2022.

[5] Diego Bustamante and Hideaki Takeda. SPARQL generation with entity pre-trained GPT for KG question answering. *CoRR*, abs/2402.00969, 2024.

[6] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code. *CoRR*, abs/2107.03374, 2021.

[7] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine, editors, *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023.

[8] Papa Abdou Karim Karou Diallo, Samuel Reyd, and Amal Zouaq. A comprehensive evaluation of neural SPARQL query generation from natural language questions. *IEEE Access*, 12:125057–125078, 2024.

[9] Alishiba Dsouza, Nicolas Tempelmeier, Ran Yu, Simon Gottschalk, and Elena Demidova. Worldkg: A world-scale geographic knowledge graph. In Gianluca Demartini, Guido Zuccon, J. Shane Culpepper, Zi Huang, and Hanghang Tong, editors, *CIKM '21: The 30th ACM International Conference on Information and Knowledge Management, Virtual Event, Queensland, Australia, November 1 - 5, 2021*, pages 4475–4484. ACM, 2021.

[10] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurélien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Rozière, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina

Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Graeme Nail, Grégoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel M. Kloumann, Ishan Misra, Ivan Evtimov, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, and et al. The llama 3 herd of models. *CoRR*, abs/2407.21783, 2024.

[11] Paolo Ferragina and Ugo Scaiella. TAGME: on-the-fly annotation of short text fragments (by wikipedia entities). In Jimmy X. Huang, Nick Koudas, Gareth J. F. Jones, Xindong Wu, Kevyn Collins-Thompson, and Aijun An, editors, *Proceedings of the 19th ACM Conference on Information and Knowledge Management, CIKM 2010, Toronto, Ontario, Canada, October 26-30, 2010*, pages 1625–1628. ACM, 2010.

[12] Ralf Hartmut Güting. Graphdb: Modeling and querying graphs in databases. In Jorge B. Bocca, Matthias Jarke, and Carlo Zaniolo, editors, *VLDB'94, Proceedings of 20th International Conference on Very Large Data Bases, September 12-15, 1994, Santiago de Chile, Chile*, pages 297–308. Morgan Kaufmann, 1994.

[13] Ehsan Hamzei, Martin Tomko, and Stephan Winter. Translating place-related questions to geosparql queries. In Frédérique Laforest, Raphaël Troncy, Elena Simperl, Deepak Agarwal, Aristides Gionis, Ivan Herman, and Lionel Médini, editors, *WWW '22: The ACM Web Conference 2022, Virtual Event, Lyon, France, April 25 - 29, 2022*, pages 902–911. ACM, 2022.

[14] Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding, 2016.

[15] Johannes Hoffart, Fabian M. Suchanek, Klaus Berberich, and Gerhard Weikum. YAGO2: A spatially and temporally enhanced knowledge base from wikipedia. *Artif. Intell.*, 194:28–61, 2013.

[16] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022.

[17] Krzysztof Janowicz, Pascal Hitzler, Wenwen Li, Dean Rehberger, Mark Schildhauer, Rui Zhu, Cogan Shimizu, Colby K. Fisher, Ling Cai, Gengchen Mai, Joseph Zalewski, Lu Zhou, Shirly Stephen, Seila Gonzalez Estrecha, Bryce D. Mecum, Anna Lopez-Carr, Andrew Schroeder, Dave Smith, Dawn J. Wright, Sizhe Wang, Yuanyuan Tian, Zilong Liu, Meilin Shi, Anthony D'Onofrio, Zhining Gu, and Kitty Currier. Know, know where, knowwheregraph: A densely connected, cross-domain knowledge graph and geo-enrichment service stack for applications in environmental intelligence. *AI Mag.*, 43(1):30–39, 2022.

[18] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de Las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7b. *CoRR*, abs/2310.06825, 2023.

[19] Nikolaos Karalis, Georgios M. Mandilaras, and Manolis Koubarakis. Extending the YAGO2 knowledge graph with precise geospatial knowledge. In Chiara Ghidini, Olaf Hartig, Maria Maleshkova, Vojtech Svátek, Isabel F. Cruz, Aidan Hogan, Jie Song, Maxime Lefrançois, and Fabien Gandon, editors, *The Semantic Web - ISWC 2019 - 18th International Semantic Web Conference, Auckland, New Zealand, October 26-30, 2019, Proceedings, Part II*, volume 11779 of *Lecture Notes in Computer Science*, pages 181–197. Springer, 2019.

[20] Sergios-Anestis Kefalidis, Dharmen Punjani, Eleni Tsalapati, Konstantinos Plas, Maria-Aggeliki Pollali, Pierre Maret, and Manolis Koubarakis. The question answering system geoqa2 and a new benchmark for its evaluation. *International Journal of Applied Earth Observation and Geoinformation*, 134:104203, 2024.

[21] Sergios-Anestis Kefalidis, Dharmen Punjani, Eleni Tsalapati, Konstantinos Plas, Mariangela Pollali, Michail Mitsios, Myrto Tsokanaridou, Manolis Koubarakis, and Pierre Maret. Benchmarking geospatial question answering engines using the dataset geoquestions1089. In Terry R. Payne, Valentina Presutti, Guilin Qi, María Poveda-Villalón, Giorgos Stoilos, Laura Hollink, Zoi Kaoudi, Gong Cheng, and Juanzi Li, editors, *The Semantic Web - ISWC 2023 - 22nd International Semantic Web Conference, Athens,*

Greece, November 6-10, 2023, Proceedings, Part II, volume 14266 of *Lecture Notes in Computer Science*, pages 266–284. Springer, 2023.

[22] Manolis Koubarakis and Kostis Kyzirakos. Modeling and querying metadata in the semantic sensor web: The model strdf and the query language stsparql. In Lora Aroyo, Grigoris Antoniou, Eero Hyvönen, Annette ten Teije, Heiner Stuckenschmidt, Liliana Cabral, and Tania Tudorache, editors, *The Semantic Web: Research and Applications, 7th Extended Semantic Web Conference, ESWC 2010, Heraklion, Crete, Greece, May 30 - June 3, 2010, Proceedings, Part I*, volume 6088 of *Lecture Notes in Computer Science*, pages 425–439. Springer, 2010.

[23] Liubov Kovriguina, Roman Teucher, Daniil Radyush, and Dmitry Mouromtsev. SPARQLGEN: one-shot prompt-based approach for SPARQL query generation. In Neha Keshan, Sebastian Neumaier, Anna Lisa Gentile, and Sahar Vahdati, editors, *Proceedings of the Posters and Demo Track of the 19th International Conference on Semantic Systems co-located with 19th International Conference on Semantic Systems (SEMANTiCS 2023), Leipzing, Germany, September 20 to 22, 2023*, volume 3526 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2023.

[24] Yixin Liu, Kejian Shi, Katherine He, Longtian Ye, Alexander R. Fabbri, Pengfei Liu, Dragomir Radev, and Arman Cohan. On learning to summarize with large language models as references. In Kevin Duh, Helena Gómez-Adorno, and Steven Bethard, editors, *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), NAACL 2024, Mexico City, Mexico, June 16-21, 2024*, pages 8647–8664. Association for Computational Linguistics, 2024.

[25] Haoran Luo, Haihong E, Zichen Tang, Shiyao Peng, Yikai Guo, Wentai Zhang, Chenghao Ma, Guanting Dong, Meina Song, Wei Lin, Yifan Zhu, and Anh Tuan Luu. Chatkbqa: A generate-then-retrieve framework for knowledge base question answering with fine-tuned large language models, 2024.

[26] Chiara Pratola Simone Tilia Sergios-Anestis Kefalidis Konstantinos Plas Mariangela Pollali Eleni Tsalapati Myrto Tsokanaridou Manolis Koubarakis Kai Norman Clasen Leonard Hackel Jakob Hackstein Gencer Sumbul Begüm Demir Marco Corsi, Giorgio Pasquali and Nicolas Longépé. Da4dte: Developing a digital assistant for satellite data archives, November 2023.

[27] Jordan Meadows, Marco Valentino, and André Freitas. Generating mathematical derivations with large language models. *CoRR*, abs/2307.09998, 2023.

[28] Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, Pouya Tafti, Léonard Hussenot, Aakanksha Chowdhery, Adam Roberts, Aditya Barua, Alex Botev, Alex Castro-Ros, Ambrose Slone, Amélie Héliou, Andrea Tacchetti, Anna Bulanova, Antonia Paterson, Beth Tsai, Bobak Shahriari, Charline Le Lan, Christopher A. Choquette-Choo, Clément Crepy, Daniel Cer, Daphne Ippolito, David Reid, Elena Buchatskaya, Eric Ni, Eric Noland, Geng Yan, George Tucker, George-Cristian Muraru, Grigory Rozhdestvenskiy, Henryk Michalewski, Ian Tenney, Ivan Grishchenko, Jacob Austin, James Keeling, Jane Labanowski, Jean-Baptiste Lespiau, Jeff Stanway, Jenny Brennan, Jeremy Chen, Johan Ferret, Justin Chiu, and et al. Gemma: Open models based on gemini research and technology. *CoRR*, abs/2403.08295, 2024.

[29] Matthew Perry Frans Knibbe-Simon J.D. Cox Joseph Abhayaratna Mathias Bonduel Paul J. Cripps Krzysztof Janowicz Nicholas J. Car, Timo Homburg. Ogc geosparql - a geographic query language for rdf data. In *Open Geospatial Consortium (OGC)*, 2024.

[30] OpenAI. GPT-4 technical report. *CoRR*, abs/2303.08774, 2023.

[31] Francesco Piccinno and Paolo Ferragina. From tagme to WAT: a new entity annotator. In David Carmel, Ming-Wei Chang, Evgeniy Gabrilovich, Bo-June Paul Hsu, and Kuansan Wang, editors, *ERD'14, Proceedings of the First ACM International Workshop on Entity Recognition & Disambiguation, July 11, 2014, Gold Coast, Queensland, Australia*, pages 55–62. ACM, 2014.

[32] Dmitrii Pliukhin, Daniil Radyush, Liubov Kovriguina, and Dmitry Mouromtsev. Improving subgraph extraction algorithms for one-shot sparql query generation with large language models. In *ISWC 2023: Scholarly QALD Challenge, November 6-10, 2023, Athens, Greece*, volume 3592 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2023.

[33] Dharmen Punjani, Markos Iliakis, Theodoros Stefou, Kuldeep Singh, Andreas Both, Manolis Koubarakis, Iosif Angelidis, Konstantina Bereta, Themis Beris, Dimitris Bilidas, Theofilos Ioannidis, Nikolaos Karalis, Christoph Lange, Despina-Athanasia Pantazi, Christos Papaloukas, and Georgios Stamoulis. Template-based question answering over linked geospatial data. *CoRR*, abs/2007.07060, 2020.

[34] Dharmen Punjani, Sergios-Anestis Kefalidis, Konstantinos Plas, Eleni Tsalapati, Manolis Koubarakis, and Pierre Maret. The question answering system geoqa2. In *2nd International Workshop on Geospatial Knowledge Graphs and GeoAI: Methods, Models, and Resources*, 2023.

[35] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21:140:1–140:67, 2020.

[36] Julio Cesar Rangel Reyes, Tarcisio Mendes de Farias, Ana Claudia Sima, and Norio Kobayashi. SPARQL generation: an analysis on fine-tuning openllama for question answering over a life science knowledge graph. *CoRR*, abs/2402.04627, 2024.

[37] Md. Rashad Al Hasan Rony, Uttam Kumar, Roman Teucher, Liubov Kovriguina, and Jens Lehmann. SGPT: A generative approach for SPARQL query generation from natural language questions. *IEEE Access*, 10:70712–70723, 2022.

[38] Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton-Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. Code llama: Open foundation models for code. *CoRR*, abs/2308.12950, 2023.

[39] Pranab Sahoo, Ayush Kumar Singh, Sriparna Saha, Vinija Jain, Samrat Mondal, and Aman Chadha. A systematic survey of prompt engineering in large language models: Techniques and applications. *CoRR*, abs/2402.07927, 2024.

[40] Melanie Sclar, Yejin Choi, Yulia Tsvetkov, and Alane Suhr. Quantifying language models' sensitivity to spurious features in prompt design or: How I learned to start worrying about prompt formatting. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024.

[41] Tommaso Soru, Edgard Marx, André Valdestilhas, Diego Esteves, Diego Moussallem, and Gustavo Publio. Neural machine translation for query construction and composition. *CoRR*, abs/1806.10478, 2018.

[42] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton-Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurélien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models. *CoRR*, abs/2307.09288, 2023.

[43] Johannes M. van Hulst, Faegheh Hasibi, Koen Dercksen, Krisztian Balog, and Arjen P. de Vries. REL: an entity linker standing on the shoulders of giants. In Jimmy X. Huang, Yi Chang, Xueqi Cheng, Jaap Kamps, Vanessa Murdock, Ji-Rong Wen, and Yiqun Liu, editors, *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval, SIGIR 2020, Virtual Event, China, July 25-30, 2020*, pages 2197–2200. ACM, 2020.

[44] Denny Vrandecic and Markus Krötzsch. Wikidata: a free collaborative knowledgebase. *Commun. ACM*, 57(10):78–85, 2014.

[45] Shujun Wang, Jie Jiao, Yuhan Li, Xiaowang Zhang, and Zhiyong Feng. Answering questions over RDF by neural machine translating. In Kerry L. Taylor, Rafael S. Gonçalves, Freddy Lécué, and Jun Yan, editors, *Proceedings of the ISWC 2020 Demos and Industry Tracks: From Novel Ideas to Industrial Practice co-located with 19th International Semantic Web Conference (ISWC 2020), Globally online, November 1-6, 2020 (UTC)*, volume 2721 of *CEUR Workshop Proceedings*, pages 189–194. CEUR-WS.org, 2020.

[46] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural*

Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022, 2022.

[47] Lingling Xu, Haoran Xie, Si-Zhao Joe Qin, Xiaohui Tao, and Fu Lee Wang. Parameter-efficient fine-tuning methods for pretrained language models: A critical review and assessment. *CoRR*, abs/2312.12148, 2023.

[48] Hongbin Ye, Tong Liu, Aijia Zhang, Wei Hua, and Weiqiang Jia. Cognitive mirage: A review of hallucinations in large language models. *CoRR*, abs/2309.06794, 2023.

[49] Wenhao Zhu, Hongyi Liu, Qingxiu Dong, Jingjing Xu, Shujian Huang, Lingpeng Kong, Jiajun Chen, and Lei Li. Multilingual machine translation with large language models: Empirical results and analysis. In Kevin Duh, Helena Gómez-Adorno, and Steven Bethard, editors, *Findings of the Association for Computational Linguistics: NAACL 2024, Mexico City, Mexico, June 16-21, 2024*, pages 2765–2781. Association for Computational Linguistics, 2024.

[50] Xiaohu Zhu, Qian Li, Lizhen Cui, and Yongkang Liu. Large language model enhanced text-to-sql generation: A survey, 2024.