Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

LLMs, Question Answering and Knowledge Graphs

Ritish Garg, BAI

A Report

Presented to the University of Dublin, Trinity College

in partial fulfilment of the requirements for the degree of

BA(Mod) in Computer Engineering (School of Computer Science & Statistics and School of Engineering)

Supervisor: Tim Fernando

April 2025

LLMs, Question Answering and Knowledge Graphs

Ritish Garg, BA(Mod) in Computer Engineering

University of Dublin, Trinity College, 2025

Supervisor: Tim Fernando

ABSTRACT

Large Language Models (LLMs) have become increasingly prevalent across industry and academia. However, despite their widespread adoption, the internal reasoning processes of these models remain largely opaque. This project investigates the alignment between the language output of LLMs and the underlying thought processes that generate those outputs. The objective is to assess whether the responses produced by LLMs exhibit coherence not only in fluent language but also in reasoning. To evaluate this alignment, the study uses Knowledge Graphs (KGs), structured factual data sources, as an external benchmark for reasoning. Natural language questions are converted into SPARQL queries using two approaches: one directly through an LLM, and another via a dedicated Knowledge Graph Question Answering (KGQA) system. By comparing SPARQL queries, their results, and other aspects, the project assesses the consistency between the natural language response of the LLM and the structured query logic. This dual-query methodology enables a heuristic evaluation of both language-thought alignment and response reliability. The findings are intended to contribute to a deeper understanding of how LLMs reliably reason, offering insight into their potential and limitations in tasks that require factual grounding and logical consistency.

Acknowledgments

Thank you Mum & Dad.

Thank you college professors.

Ritish Garg

University of Dublin, Trinity College
April 2025

Contents

Abstract **i**

Acknowledgments **ii**

Chapter 1 Introduction **1**

1.1 Project Overview 1

1.2 Report Structure 2

Chapter 2 State of the Art **3**

2.1 Background 3

2.2 Closely-Related Work 5

2.2.1 Language-Thought Alignment in LLMs 5

2.2.2 Structured Reasoning in KGQA Systems 6

2.3 Summary 6

Chapter 3 Design **7**

3.1 Problem Formulation 7

3.1.1 Identified Challenges 8

3.1.2 Proposed Work 9

3.2 Overview of the Design 10

3.3 Summary 11

Chapter 4 Implementation **12**

4.1 Description of the Solution 12

4.2 Implementation Issues 20

4.3 Summary 21

Chapter 5 Evaluation **22**

5.1 Results 23

5.2 Summary 25

Chapter 6 Conclusions & Future Work **26**
6.1 Conclusions 26
6.2 Future Work 27

Bibliography **29**

List of Figures

5.1 LLM Output for Natural Language question 23
5.2 LLM Training Loss 24
5.3 LLM Evaluation Training Loss 24
5.4 LLM Validation Loss 25

Chapter 1

Introduction

1.1 Project Overview

The whole project is about the belief that 'Assessing whether LLM-generated answers align with their underlying reasoning, using Knowledge Graphs[1, 2] to support judgment of alignment and reliability' and arguments in its support. There is much more to understand than just reading the statement. The belief essentially checks if the use of LLMs is evidently beneficial, without being hallucinating, through fluency checking in both language and thinking thus, implying alignment between them. Although it is known and there is a claim that good fluency in language is not equal to good fluency in thinking or reasoning, and vice versa[3]. In simpler terms, the problem statement is whether LLMs really say or do something that they thought of, and vice versa, or are just making new information like hallucinations which are factually incorrect. The procedure used to approach this problem is to take advantage of already existing structured data sources or knowledge stores, Knowledge Graphs(KG)[1]. These will be used as a reference point for ground truths in the reasoning of LLMs. There are precisely two parts to the approach, thought of as a solution: LLM and KG part accurately, Knowledge Graph Question Answering System(KGQA)[2, 4]. LLMs are themselves a type of Question Answering(QA) system but not a KGQA system. A question answering system is an interface to ask questions and get correct answers to them which LLMs today are capable of. However, there are important problems, such as hallucinations, that need to be solved. Both components of the approach use SPARQL Protocol And RDF Query Language(SPARQL)[5], KG query language, as their basis to work further. Since both of these are AI systems, the hope is to pose a Natural Language(NL) query as input and get the corresponding SPARQL query as output. In addition, it is important to understand[4] the question asked by the user to LLM and KGQA. This understanding along with some SPARQL-query-related heuristic will help in the evaluation of the output, which will be seen later in the report along with more information in detail. In combination, LLMs integrated with KG is an emerging field of research with prominent examples such as Graph-Based Retrieval Augmented Generation (GraphRAG)[6]. The idea of domain-specific KG, such as in medicine, cyber-security, gives motivation for further work in KG and, thus, integration with LLM in the form of Retrieval Augmented Generation(RAG)[7], for example.

1.2 Report Structure

The report starts with an overview of the project that revolves around LLMs, KG, and their integration to solve and assess the claims made about LLMs for their authenticity in thought and thus coherence with their language usage, for better understanding, in the broad realm of Question Answering Systems (QA). It then transitions to the chapter of related work or the state-of-the-art (SOTA), if present yet, in the field of the project's problem statement and similar. This addresses in a large part what KGQA, LLM, and the cognitive ability of LLM[3] is, which is also the belief to be assessed. This will help to understand what is being followed to solve the problem serving as a fundamental. Getting an overview and work done so far will help in designing the approach taken as a solution to the problem, and this is discussed in the Design chapter 3. This mainly contains the essential components found while formulating the problem, the identified challenges and proposed solutions, and the overall methodology of the approach designed. Designed solution with confidence can then allow one to implement it. The implementation details are discussed in the Implementation chapter which essentially contains the formal solution with an overview and nuanced descriptions. The results and findings are presented in the Evaluation chapter 5 with comparisons of existing work, if present. It contains any experimentation details if done and helps in a better understanding of the project in the respective research field. Basically, it summarizes all the theoretical and empirical results found. The essence of the whole project is concluded in the Conclusions and Future Work chapter 6 having the summarized important aspects of the project and any future work that can be and is intended to be done and specifically, in relation to the respective research field.

Chapter 2

State of the Art

The following chapter discusses the background of the project in the specific research field along with any closely related works or work present in similar fields done, if present yet. It is divided into two sections as mentioned, with the background discussing the motivation, problem statement, and relation to the research area the project is from. It will also help in better understanding the project with the descriptions of the existing technologies[4] used as components to the problem's solution and form basis of the work presented in the report. The other section mainly discusses the work done in the specific project-related area[3] within the broad research area. It may also have descriptions of any SOTA present with respect to the problem statement. The specific project area has yet to gain much attention with the growing use of Knowledge Graphs(KG) in Retrieval Augmented Generation(RAG)-based applications[7, 6]. There is a broad scope for future work in the same area.

2.1 Background

The specific area project deals in is at the integration of LLMs, KG and Question Answering for the purpose of examining the belief aforementioned in the Introduction and the claim that LLMs are good at language but less so, thought[3] depending on the results found. This is essential to be done in order to have reliable AI systems especially, LLMs since their widespread adoption. The foreground part of the project is about the LLMs which are the main component of the discussion since their cognitive ability is being questioned. This is the foremost reason behind the motivation of project as well. The only concept that makes these LLMs comparable to the Human Brain is the adoption of the idea of how neurons may interact with each other and that is yet not understood completely. Thus, it is known for a fact that LLMs have learned the formalism of language and are good at it convincingly. But as with all the Deep Learning architectures, it is not yet known how do these systems actually work internally ? This implies the question for LLMs about their thinking ability and whether the belief is actually true or not. In simple terms, checking if their language and thoughts really align with each other. Since LLMs are precisely Question Answering(QA) systems trained on a large corpora of text, it becomes important for the outputs generated by these systems to be correct and not be just made up information i.e., hallucinations. QA systems are interfaces on machine which can act as an assistant helping the user with their questions and queries of any kind or domain-specific. To support all these ideas, Knowledge Graphs[1] have been chosen as part of the solution approach. KG are structured sources of data with semantics i.e., Knowledge which is in the form of a Graph data structure. A Graph data structure is a nodal structure with nodes and edges, which connect nodes. KG collectively are structures having meaning with the nodes representing different entities within the real world and edges representing the relations between these entities. KG can be open-domain such as DBpedia, Wikidata or domain-specific such as specifically for medicine and cyber-security. DBpedia has been chosen as the KG for this project. All these KG have their dedicated, specific endpoints to infer or precisely, to query them and the language used for querying is SPARQL[5] which is executed over these endpoints[8]. There can be various wrappers possible in libraries around these endpoints such as SPARQLWrapper in python language. SPARQL as a query language is used specifically for Resource Description Framework(RDF)[9] based structures which is a way of representing knowledge in graphs. Another important aspect to understand after demystifying KG and QA is of Knowledge Graph based Question Answering(KGQA)[4, 2] for project understanding. KGQA has been attended to for a considerable amount of time with many different approaches[4, 10, 11] designed and published. It can prove as a medium of trusted information whether domain-specific or not. Since there are many approaches to them, the fundamental approach to the current SOTA in KGQA can be decomposed, but not limited to, into five stages typically– Question Analysis, Phrase Mapping, Disambiguation, Query Construction and Distributed Knowledge[4]. All these phases have their own purpose and method of working, each with several techniques developed for over a period of time[2]. However, KGQA does come with its own set of challenges, even for each phase. A KGQA system being a QA system itself, it is stated in [4],

> In order to address these challenges, QA systems generally combine techniques from natural language processing, information retrieval, machine learning, and Semantic Web.

In general, these are the technologies that form the basis of the project, as well as its requirements with an additional concept of fine-tuning[12] and prompt engineering. These terms are used with respect to LLM training and inference. Pre-trained LLM[13, 14, 12] are used in the project which has been fine-tuned for the specific use case of conversion of Natural Language questions or queries to corresponding SPARQL queries. Several models have been from [14] and [12] before choosing the final LLM to be used due to issues such as data formats, different fine-tuning techniques and similar. There are several ways of fine-tuning LLM such as parameter-efficient fine-tuning(PEFT) with LoRA, QLoRA techniques[15], and dedicated libraries such as Unsloth[12]. There is much more detail to fine-tuning than can be discussed here. Precisely, it is a way of using the general pre-trained LLMs for the specific use case under consideration. However, with prompt engineering, the accuracy of the LLM for the conversion use case can be increased, although prompt engineering alone may also help in the conversion with some examples given in the prompt itself. In general, there will be a need for some prompt engineering to structure the best required outputs and their specific formats, SPARQL and its programmatic rules for this project. The working of these systems, whether LLMs or KGQA is highly influenced by the understanding depth including pragmatics of the natural language queries posed to them, and it remains a concern with current work done in the area. In conclusion, LLM, KG, QA systems integrate together to form a robust AI system which understands, and the goal of this project is to assess its behavior in language and thought if such a system can be well built and is existential.

2.2 Closely-Related Work

To truly understand whether LLMs "think" about what they say, it helps to step back and look at what others have already explored in this space. Instead of diving into individual papers one by one, this section focuses on two key ideas that directly relate to the core of this project. The first is about how LLMs use language and whether fluency actually means they understand what they're saying[3]. The second looks at systems built around Knowledge Graphs, which are designed to answer questions in a more structured, transparent way[4, 2]. These two lines of work offer different but equally important perspectives that help shape the motivation and approach for everything that follows.

2.2.1 Language-Thought Alignment in LLMs

A recent paper[3] by Mahowald et al. (2024) explores an important idea: just because a model is good with language does not mean it is actually reasoning. The authors split language skills into two types, formal linguistic competence, which deals with grammar and sentence structure, and functional linguistic competence, which is about using language meaningfully in real situations. According to their findings, LLMs tend to do quite well with formal language—they can write fluent, grammatically correct sentences. But when it comes to deeper tasks, such as reasoning through a problem or understanding context, their performance is less consistent. Often, they need additional support, such as fine-tuning or external tools, to perform well in these areas. This connects directly with the main focus of this project: that fluency alone is not enough to consider a model to be reliable. What really matters is whether there is actually a reasoning behind what it says.

2.2.2 Structured Reasoning in KGQA Systems

In contrast to the statistical generation of language by LLMs, Knowledge Graph-Based Question Answering Systems (KGQA) are built to answer questions by directly querying structured knowledge bases or sources[16]. Diefenbach et al. (2018) present a detailed survey of QA systems over knowledge bases[4], identifying techniques for key tasks such as natural language interpretation, entity disambiguation, SPARQL query construction, and execution over distributed datasets. These systems provide high factual reliability and transparent reasoning paths, making them suitable benchmarks against which LLM-generated reasoning can be compared. Although these systems also come with their own set of challenges.

2.3 Summary

This chapter presented the foundational concepts and research supporting the project. It began with the project background overview and the technologies involved-LLMs, Knowledge Graphs, QA systems, and fine-tuning techniques. The closely related work section explored two main ideas: the gap between language fluency and reasoning in LLM, and the structured, fact-based approach of KGQA systems. Together, these form the basis for the project approach to compare the output of the LLM and KGQA systems to evaluate their reasoning alignment and reliability together with some other mechanisms considered for evaluation[17].

Chapter 3

Design

This chapter presents the overall design and approach adopted to address the core problem introduced earlier. It builds on the background and related work discussed in previous chapters and outlines how the proposed solution takes shape. The chapter begins by refining the problem formulation and highlighting key challenges, followed by a description of the proposed work. Then it introduces the overall design of the system that brings together LLMs and Knowledge Graph-based Question Answering Systems (KGQA) to explore the alignment between language and reasoning.

3.1 Problem Formulation

The problem under consideration is very well mentioned in the previous chapters. However, there can be some details worth attending to and is done in this section. The main aim of this project is to assess whether the claim by [3] paper is actually true which says that LLMs are good at language but less so, thought. This assessment will signal towards the reliability of LLMs as the consequence, being the second question posed in this project. In simple terms, good fluency in language does not indicate good thinking even though it is convincingly sound and vice-versa. However, it can be demystified when taken in perspective with respect to to Human speech-thought alignment since all of Artificial Intelligence has taken inspiration from Humans and specifically, the Brain. The posed questions are important to work for since the fundamentals of communication and specifically, Understanding or correct interpretation in a conversation is an essential part of being a Human to survive. With advancements, the humankind is progressively moving towards Humanoids, Artificial General Intelligence(AGI) for which to happen, LLMs as a foundation have to possess the questioned necessities in this project. Another aspect that is of importance is whether the information LLMs provide to a posed NL query is correct or not. Only being aligned is not sufficient. The case should be as is with Humans that an AI assistant, in general and an LLM, in particular should respond to a question naturally i.e., to answer if it knows concretely and vice-versa or to discuss/reason with the question poser but should not Hallucinate, in any case. In addition, the significant concern is Question Understanding(QU) in every aspect whether syntax, semantics, pragmatics of the natural language question posed to be able to answer. This part is crucial for assessing the correctness of the answer plausibly signaling towards not hallucinating. However, there is no concrete evidence that QU alone will solve the project's problem. This can establish the answer in the language being unverified correct but not the thought process behind that. The approach to get the thought process i.e., the internals going in the LLM such as Chain-of-Thought(CoT) is not explicitly studied but is attended to with some convincing solutions. This is considered because not all LLMs provide their reasoning step-by-step as is in CoT or similar techniques. These reasoning understanding-and-evaluation solutions can indicate if it matches the answer in the language given with the context of the reasoning being of utmost importance. This is done by involving some attention mechanism[17] to understand that context discussed in the subsequent sections and chapters. Another major concern is the question type, expected answer type, and whether KGQA[4, 2] is able to query the KG for complex question types which involve multi-hop reasoning and similar techniques. All this analysis of the problem is verified by using KG[1] and KGQA[2]. Although KGQA is an old, separate field of research itself, it is leveraged in this project. KG contain verified knowledge which can be from either domain-specific graphs or open-domain depending on the use case and severity of the knowledge. KGQA are the systems which help in querying and accessing these KG. Although, KGQA have their own set of challenges such as problems within each of its five stages like entity disambiguation, entity resolution, relation linking, missing information in the KG itself and so on[4], these will not be addressed in this project but can probably be discussed. Additionally, there can be some more nuanced problems within the project not thought off and encountered yet. Also, reasoning with LLMs and KG for QA[18] is yet another area of work involving slightly different concepts which will be discussed about in Further Work chapter 6. In conclusion, the aforementioned formulation and analysis of the problem statement of the project is approached to solve and hence, the further sections with details.

3.1.1 Identified Challenges

There are always challenges with any problem statement, whether based on existing work or implementations of the project. This section discusses the challenges or gaps within the existing work, if any, and the corresponding solutions in the next section. However, with the amount of existing work reviewed for this project, there are not many nuanced challenges identified. Moreover, the fundamental challenge for LLMs, which is being mitigated with further research, is to make them understand language in different contexts with the ability to interpret correctly. However, it is not a challenge for this project since LLMs pre-trained on a large corpora of text have been used. Specifically, the instruct versions of these LLMs[13] that are fine-tuned for conversation style use case as is with an assistant. Another challenge is how to get these LLMs to specifically output the corresponding SPARQL queries for the natural language question. This identification and its solution will help narrow down the LLM work, and hence increase the accuracy and performance. In general, it is also a challenge to ensure that the LLM does not hallucinate, i.e., to come up with new hypothetical information. This is a complex one for which not enough work was explored in this project but is found in the literature. In addition, the ability of LLMs to reason concretely has been discussed and approached in the report with a clear view of the challenges mentioned above, where possible. In addition to these, the main challenges that arise while working with the report's project area are the building of KGQA systems[2]. This is due to the fact that QA over KG is a separate area in QA systems with a lot of research [4, 2], and thus approaches in the literature. It is difficult to implement these with novel approaches due to time constraints. This challenge is not addressed in the report. This can be understood with the discussion in the previous chapters and sections. Consequently, these challenges will be approached in the following sections with nuances as encountered.

3.1.2 Proposed Work

This section discusses the proposed approaches for the challenges identified above and the problem formulated, in general. The work consists of an LLM and a KGQA system. KGQA is used as a mediator to query KG for which it needs to be built from scratch[4] or to use existing approaches to them[10, 11]. Both of these systems take inputs in natural language and outputs in slightly different forms, including the SPARQL[5] query. The reasoning for producing the SPARQL query is also outputted by the LLM, if possible. These SPARQL queries are used for the evaluation of understanding of the question by LLM specifically in some way, considering the KGQA approach used along with some human oversight as well. Here only, the heuristic-based comparison will be done as mentioned earlier. These queries are executed[8] to assess the accuracy of the answers with respect to pairs of question-answer type, and to form a judgment about hallucinations. The comparison of SPARQL queries is done for better observations, and thus inferences as mentioned. There can be a similarity threshold (based on heuristics) for the comparison-based assessment of the SPARQL query. The KG part in the entire setting will act as the verifier of the understanding and output of the LLM. These will help to form results with respect to the correctness & hallucinations, question understanding(QU) to form the correct SPARQL query and using the context of the reasoning applied to form the SPARQL query, assessing the LLM QU, language, and therefore the judgment for the alignment of the LLM language and thought. This in turn implies the question of reliability of LLMs with the assessments mentioned. As a result, the objectives and goals of the project are met with evaluations and inferences made. This is an overview of the work with a detailed description of the solution design in the following section.

3.2 Overview of the Design

This section provides the detailed description of the approach designed and followed for the problem statement of the project. The approach consists of two parts: the LLM part and the KGQA part. The LLM component is the main part under consideration with the KGQA component being the verifier of it. Both the systems are posed with Natural Language questions or queries and their task is to completely understand the question and do the respective work discussed in the following text. The operation is to convert the Natural Language question to its corresponding SPARQL[5] query. For the LLM to do this, it is expected for it to provide the reasoning behind along with the SPARQL query. The KGQA has a query construction stage in its working process[4] from where the SPARQL query will be taken. After having the SPARQL queries generated from both the systems, it is helpful how LLMs get to produce SPARQL queries. An LLM[13] is fine-tuned[19] specifically for the Natural Language-to-SPARQL query conversion task. A dataset[20] from HuggingFace has been used to fine tune the LLM which contains NL-SPARQL correspondence. The dataset used can be a challenge for the accuracy and performance of the LLM for this task due to the constraints such as data quality, dataset size. Getting the SPARQL queries ready leads to an assessment for the question type or intent and answer type correspondence check implying the LLM understands well the required initials that are verified by the KGQA. This also assesses the correctness of the LLM SPARQL query if the execution[8] results match or are nearby to each other in the space when verified with the KGQA results, but just by the answers. The SPARQL queries will be compared separately as well taking in perspective the context of the question and its understanding using which some single real number metric will be designed with thresholding for their correct correspondence measure. This implies the language being correct and not hallucinated i.e., LLMs do have good fluency and correct, required output with respect to the question. The thinking and reasoning is yet to be assessed. It is done by using the reasoning given by the LLM for the SPARQL generation to understand it completely using some attention mechanism[17] and then, comparing that context with the NL-SPARQL generation correspondence to evaluate if the thinking really matches the language outputted by the LLM. KGQA is used in comparing the question-answer-type pairs, generated SPARQL queries and whether the entities & relations used by LLM output actually exist[1] in the real world. The major problem here is the missing information in the KG itself. This problem is not addressed here. After the language and reasoning of the LLMs is evidently shown to be aligning with verification using KG and KGQA, the belief and hence, the claim[3] stands false yet true. This is because it may not be false for every case. The claim itself was of good at language but less so, thought. Given the research done in the LLMs field, the problem with complex questions stands still which tells that the LLM struggles with them at forming SPARQL queries.Hence, the language fluency may stumble there but not completely again even though the reasoning is seemingly well. However, this can convincingly indicate the LLMs being reliable to some information and not for other. In conclusion, this is the approach thought off and hence, the design. The results of implementing the approach will actually tell if it works or not. In addition, the approach could not be implemented completely due to time constraints and therefore, has a scope of future work as discussed in the Conclusions & Further Work chapter 6. The following summarizes the design and methodology of the project and leading to the implementation chapter.

3.3 Summary

This chapter outlines the plan to check if LLMs truly understand language or just sound good, and if that affects how much they can be trusted. The approach uses LLMs and Knowledge Graphs (KG), which are like organized fact databases. Both will get the same questions. The LLM will try to answer and explain its reasoning by translating the question into the KG query language (SPARQL). Then a comparison of the LLM translation and reasoning with how the KGQA system understands the question and the actual KG answers will be done. This will help to see if the LLM really gets it and if its answers are accurate based on the KG. This chapter details how this comparison will be done to assess the LLM understanding and reliability hence, contributing the major part of the report.

Chapter 4

Implementation

This chapter deals with the implementation of the project as a whole. This mainly consists of essential code snippets with explanation, workflow of the implementation in text where applicable, and implementation issues encountered while implementing this project. Although the implementation is not complete as the design methodology 3 describes, all the implementation done is described here. It goes all the way from creating working directories in containers[21] for GPU use to accessing the LLM and dataset & using fine-tuning techniques to infer the fine-tuned LLM for the desired outputs and performing further evaluations. In conclusion, this part of the report stands as evidence of the approach designed and, hence, the results found and observations made.

4.1 Description of the Solution

The implemented solution intends to follow the approach designed in the preceding chapters 3. However, as stated earlier, it is not yet complete. There were many workflows for implementing the approach, such as trying to implement the KGQA part first and then the other part, but given the complexity of KGQA implementation, the LLM part was chosen to work with in the initial stages of implementation. The LLM part consists of the fine-tuning[19, 12] of the chosen LLM for the task of converting natural language questions posed to the correct corresponding SPARQL queries, which would be used to query the Knowledge Graph[16] used. Before going directly to the fine-tuning process directly, there are essentials to be done such as dataset & model loading, data preprocessing, data formatting with respect to LLM conversation-style formats, etc. until the actual fine-tuning is initiated. All of this will be discussed in this section. Since the LLM used is by MistralAI[13], an exclusive code base has also been developed specifically for fine-tuning the Mistral models, mistral-finetune[19]. The implementation gets the necessary libraries for accessing models & datasets, code base and its requirements into the code environment. Getting the environment ready is essential for the workflow and thus a virtual environment was created for fluent code flow without any interruption caused. Setting up the environment leads to the download of the chosen LLM model[13] in its own separate directory. Exactly, the instruct version of the LLM is chosen as in 4.1.

```
from huggingface_hub import snapshot_download
from pathlib import Path

#Set the path to the /workspace/working directory
mistral_models_path = Path('/workspace/working/', 'mistral_models')
mistral_models_path.mkdir(parents=True, exist_ok=True)

#Download model snapshot to the desired path
snapshot_download(repo_id="mistralai/Mistral-7B-Instruct-v0.3", allow_patterns=["
    params.json", "consolidated.safetensors", "tokenizer.model.v3"], local_dir=
    mistral_models_path)

#snapshot_download(repo_id="unsloth/mistral-7b-instruct-v0.3-bnb-4bit", allow_patterns
    =["config.json", "model.safetensors", "tokenizer.model.v3"], local_dir=
    mistral_models_path)

#Copy the model files to the /workspace/working directory(if necessary)
#!mv /workspace/working/mistral_models/7B-instruct-v0.3 /workspace/working/
    mistral_models
#!rm -r /workspace/working/mistral_models/7B-instruct-v0.3
```

Listing 4.1: Download of the LLM model into its own separate directory created with some, if needed, code.

Downloading the model alone is not enough. It needs data to be fine-tuned for it to perform the conversion task. So, a separate data directory has been created by performing some directory path operations. The data set[20] has been downloaded to a DataFrame using the read_parquet() function in Pandas since the dataset is in parquet format4.2.

```
import pandas as pd
import copy
df = pd.read_parquet("hf://datasets/julioc-p/Question-Sparql/data/train-00000-of
    -00001.parquet")
dftemp = df.copy()
dftemp
```

Listing 4.2: Download of the dataset for fine-tuning along with some necessary libraries. A copy of df is fomred in dftemp to avoid any operational changes affect the original dataset in df.

After downloading the LLM[13] and the data set[20], the data set must be processed before being used for the LLM fine-tuning[19]. 4.3 shows how the data preprocessing was carried out along with sanity checks and proper requirements being satisfied for what data is actually needed.

```
#Checking only English language and DBpedia Knowledge Graph data & extracting it
dftemp = dftemp[dftemp["language"] == "en"]
dftemp = dftemp[dftemp["knowledge_graphs"] == "DBpedia"]

#Sanity checks
dftemp[dftemp["language"] != "en"]
dftemp[dftemp["knowledge_graphs"] != "DBpedia"]
```

Listing 4.3: Data preprocessing for the extraction of the English NL questions and DBpedia KG queried SPARQL queries in the dataset. Some rechecking for any not needed data records after cleaning.

The new data extracted from the downloaded data set must be in accordance with the accepted format of the inputs given to fine-tuning the mistral LLM[13, 19]. The descriptions and formats can be found under the code base of mistral-finetune[19] and Unsloth[12] for a further, unrelated to mistral-finetune formats, and a deeper understanding of the conversation formats. This has to be done because it has been made strict with respect to the data formats by mistral-finetune. This is shown in 4.4. In addition, 4.5 checks for the maximum length of the input dictionary as requested by the mistral model data format to use the good enough sequence length to train, that is, fine-tune the LLM accordingly.

```
#Removing the unnecessary rows with "\n" & columns with index resetting
dftemp.reset_index(drop = True, inplace = True)
dftemp.drop(columns=["language", "knowledge_graphs"], axis = 1, inplace = True)
tbremovedrows = dftemp["sparql_query"].str.contains("\n")
dftemp = dftemp[~tbremovedrows]
dftemp.reset_index(drop = True, inplace = True)



#Code related to the LLM chat formats & data columns' name changes
dftemp["messages"] = dftemp.apply(lambda row: [{"role": "user", "content": row["
    text_query"]}, {"role": "assistant", "content": row["sparql_query"]}], axis=1)
dftemp.rename(columns = {"text_query" : "NL_Query"}, inplace = True)
dftemp.drop(columns = "sparql_query", inplace = True)
```

Listing 4.4: Continued data preprocessing for the requirements by removal and transformation of that data into the necessary chat formats as provided by mistral-finetune.

```
#Requred code & sanity check for the sequence length to be used for Fine-tuning
print(dftemp["NL_Query"].str.len().max())
print(dftemp["messages"].str.len().max())
print(
    dftemp["messages"]
    .apply(lambda x: " ".join([msg.get("content", "") for msg in x if msg.get("role")
        == "assistant"]) if isinstance(x, list) else "")
    .str.len()
    .max()
)
```

Listing 4.5: Input dictionary length check for accordingly choosing the good enough sequence length to fine-tune the LLM.

For training and fine-tuning any model, the data set must be split into two or preferably three separate datasets-train, evaluation & test. However, for this project, the dataset used has been split into two, with the evaluation used as both a train and an evaluation set. In addition, some index resetting and the split data set conversion to jsonl files[19] are performed as expected by the LLM, respectively, and can be seen in 4.6.

```
#Sanity checks for expected & aceepted data by the LLM
print(df_train.iloc[0]["NL_Query"])
df_train.iloc[0]["messages"]

#Dataset splitting
df_train=dftemp.sample(frac=0.80,random_state=200)
df_eval=dftemp.drop(df_train.index)
df_train.reset_index(drop = True, inplace = True)
df_eval.reset_index(drop = True, inplace = True)

#Saving data into .jsonl files
df_train.to_json("NL_to_SPARQL_train.jsonl", orient="records", lines=True)
df_eval.to_json("NL_to_SPARQL_eval.jsonl", orient="records", lines=True)
```

Listing 4.6: Data printing for expected format check. Splitting of the dataset into train and validation set with index resetting. At last, conversion to jsonl files as mentioned in the mistral-finetune code base requirements.

The data sets have been created but not validated yet; the configuration for fine-tuning the LLM and validating the data must be created first. The configuration consists of several steps such as the datasets, model used, LoRA hyper-parameters for parameter efficient fine tuning(PEFT)[15], optimization parameters and so on. This configuration is in the yaml file[19] which will hence be used for both data assessment for irrelevant data-formatted records, validation, and fine-tuning. All this is done according to the constraints and requirements of the mistral-finetune code base as shown in 4.7.

```
import yaml
config = """
#data
data:
  instruct_data: "/workspace/working/data/NL_to_SPARQL_train.jsonl"
  data: ""
  eval_instruct_data: "/workspace/working/data/NL_to_SPARQL_eval.jsonl"

#model
model_id_or_path: "/workspace/working/mistral_models"
lora:
  rank: 16
  scaling: 16

#optim
#tokens per training steps = batch_size x num_GPUs x seq_len
#we recommend sequence length of 32768
#If you run into memory error, you can try reduce the sequence length
seq_len: 2048
batch_size: 1
num_microbatches: 8
max_steps: 300
optim:
  lr: 1.e-5
  weight_decay: 0.1
  pct_start: 0.05

#other
seed: 0
log_freq: 10
eval_freq: 100
no_eval: False
ckpt_freq: 100

save_adapters: True #save only trained LoRA adapters. Set to `False` to merge LoRA
    adapter into the base model and save full fine-tuned model

run_dir: "outputs"
"""

#save the same file locally
with open('example/7B.yaml', 'w') as file:
    yaml.dump(yaml.safe_load(config), file)
```

Listing 4.7: Configuration setup for validation of the data and the fine-tuning of the model as described in the text. Dumped the configuration into a yaml file to be used ultimately.

The LLM is now ready to be fine-tuned after the data is validated4.8 and, hence, the fine-tuning is performed in the subsequent codes4.10.

```
#Sanity check for the existence of the configuration file
import os
print(os.path.exists("example/7B.yaml"))

#Data assessment for any incorrect data format
!python -m utils.reformat_data /workspace/working/data/NL_to_SPARQL_train.jsonl
!python -m utils.reformat_data /workspace/working/data/NL_to_SPARQL_eval.jsonl

#Data validation check for the correct data format
!python -m utils.validate_data --train_yaml example/7B.yaml
```

Listing 4.8: Check for the existence of the needed yaml file. Data assessment and validation is also performed as descriebed in the text.

Since the whole process of fine-tuning & training, in general, and inferencing is computationally expensive and resource intensive, Graphics Processing Units(GPUs) are needed. These are detected in the environment setup on the machine used by the code in 4.9.

```
#GPU checking
import torch
print("Number of GPUs Available:", torch.cuda.device_count())
for i in range(torch.cuda.device_count()):
    print(f"GPU {i}: {torch.cuda.get_device_name(i)}")
print(torch.cuda.get_device_name())
!nvidia-smi

#Detected GPU access allocated
os.environ["CUDA_DEVICE_ORDER"]="PCI_BUS_ID"
os.environ["CUDA_VISIBLE_DEVICES"]="0"
```

Listing 4.9: Checking if GPU is even available and accessible. If it is indeed so, then it is allocated or allowed-after-detected for use.

Training is performed using the command4.10 on a single GPU node as detected. As mentioned earlier, the yaml configuration file has all the requirements needed to fine-tune the LLM. In addition, all the constraints and strict requirements necessary by the [19] code base have been fulfilled.

```
!torchrun --nproc-per-node 1 -m train example/7B.yaml
```

Listing 4.10: A shell command passed to the shell by the python kernel. Runs to train the LLM on a single node using torchrun and configuration file.

Getting the fine-tuning done of the LLM leads to the directory moving and listing operations (not included here), with the outputs directory being a separate one for storing all the output files. It contains all the outputs with respect to the trained LoRA adapters and model weights. The next step is to perform inference on the fine-tuned LLM for SPARQL query conversion of the posed natural language question. For inference, MistralAI provides its own mistral_inference[22] library designed specifically for inferring mistral models. The library is installed using the pip package before using it. It is shown in 4.11.

```
#Installing the inference library
%pip install mistral_inference

#Actual inferencing performed
import safetensors
from mistral_inference.transformer import Transformer
from mistral_inference.generate import generate
from mistral_common.tokens.tokenizers.mistral import MistralTokenizer
from mistral_common.protocol.instruct.messages import UserMessage
from mistral_common.protocol.instruct.request import ChatCompletionRequest

#load tokenizer
mistral_tokenizer = MistralTokenizer.from_file("/workspace/working/outputs/checkpoints
    /checkpoint_000300/consolidated/tokenizer.model.v3")

#load model
model = Transformer.from_folder("/workspace/working/mistral_models")
model.load_lora("/workspace/working/outputs/checkpoints/checkpoint_000300/consolidated
    /lora.safetensors")

#Savng the merged model of LoRA adapters into the base model
safetensors.torch.save_model(model, "/workspace/working/outputs/finetunedmodel")

#Natural language posed with the prompt(System prompt + User prompt) shown
query = input("Any questions?\n")
prompt = f"You are a helpful AI assistant. We are working over DBpedia Knowledge Graph
    . Your task is to provide the corresponding SPARQL query for the following natural
     language query: {query}. Also, provide the reasoning and thinking you applied for
     generating that specific SPARQL query in a detailed paragraph."
```

```
messages = [{"role" : "user", "content" : prompt}]

#chat completion request
completion_request = ChatCompletionRequest(messages=messages)

#encode message
tokens = mistral_tokenizer.encode_chat_completion(completion_request).tokens

#generate results
out_tokens, _ = generate([tokens], model, max_tokens=16384, temperature=0.0, eos_id=
    mistral_tokenizer.instruct_tokenizer.tokenizer.eos_id)

#decode generated tokens
result = mistral_tokenizer.instruct_tokenizer.tokenizer.decode(out_tokens[0])
print(result)
```

Listing 4.11: Demonstration of the full inference pipeline using a fine-tuned Mistral model. It loads the tokenizer and base model, applies the LoRA adapter weights, and saves the merged model. A natural language query is taken from the user, wrapped in a structured prompt, and tokenized. The model then generates a corresponding SPARQL query along with reasoning, which is decoded and printed as the final output.

The inference done gives the required SPARQL query with the reasoning behind, and this SPARQL query is executed at the SPARQL endpoint[8] for DBpedia[16] using the wrappers in Python. The wrapper used is SPARQLWrapper. The outputs are in JSON format. This is demonstrated in 4.12.

```
from SPARQLWrapper import JSON, SPARQLWrapper
SPARQLendpt = SPARQLWrapper("http://dbpedia.org/sparql")
SPARQLendpt.setReturnFormat(JSON)
SPARQLendpt.setQuery('''LLM GENERATED SPARQL QUERY GIVEN BY LLM GOES HERE''')
LLMAnswer = SPARQLendpt.query().convert()
LLMAnswer
```

Listing 4.12: SPARQL query genereted by the LLM is executed as explained in the text and outputted in a structured human-readable JSON format.

This was the whole implementation for the LLM part, but not the complete implementation as intended. This will give the indications for the 'LLM language being correct?' and the reasoning behind alignment when done by human oversight. Empirically, it is time intensive because of the constraints on time itself. The evaluations and results are discussed in the next chapter 5 with the implementation issues encountered mentioned in the next section.

4.2 Implementation Issues

There were many issues encountered while implementing the project's designed approach in code. There were two parts to it as mentioned previously. Moreover, not only were implementation issues raised, but conceptual difficulties in understanding the fine-tuning techniques and building KGQA systems were raised. However, the issues are the following: the foremost issue is with the computational resources to run the whole project. The intensity of the project can be estimated by the fact that it worked without interruptions on an 80 GB VRAM GPU on a cloud-provided container[21]. All the other methods whether using Windows Subsystem for Linux(WSL) or manufacturer OS to use system resources or using Google Colab, along with their own either environment setup issues or other issues such as encoding-related (ASCII, UTF-8, etc.) in Colab, the implementation would not quite work well except containers. However, choosing the library or framework to fine-tune the LLM was another challenge. There are many models available, some with their specific fine-tuning setups available such as mistral models[19] while others can be fine-tuned using dedicated libraries such as Unsloth[12]. Models from Ollama[14] also have their own documentation to fine-tune them. Although each of these techniques and code bases come with their own requirements and benefits, the problem is in getting around the data formats of specific LLMs in addition to the code base-specific formats posing as a typical constraint. However, preparing data with respect to the specifics does take time, the final processed data used still is not perfect and can be transformed further with possibly increasing the accuracy of the LLM. Consequently, inference became difficult even after fine-tuning was performed well on the same resources due to the computational resource constraint. The reason behind this was that the code would load the whole base model and its trained LoRA adapters combined into the GPU VRAM which being small could not load the model and hence, inference could not be done. However, distributed inference was tried on GPUs available to resolve the issue but the out-of-memory(OOM) errors persisted. So, the Issues page[23] on Github was referred for its solution but did not help. However, this issue was resolved with the use of GPU A800 PCIE GPU with 80 GB VRAM i.e., an unexpectedly large VRAM. Since the LLM part alone appeared to have many errors and issues, the KGQA part was supposed to have many more issues both execution-wise and conceptually. This can be said because the KGQA part was started initially, but because of many approaches to each stage of building a KGQA system, it was hard to build one in the time constraints provided. For reference, one concept is of Named Entity Recognition(NER) which alone has many approaches in the literature and industry provided with the conceptual difficulty of each. In conclusion, there were and will be many implementation issues that come up for further work to be done that need attention with good conceptual understanding and proper literature reviews of previous research done, specifically in the KGQA systems research area.

4.3 Summary

This chapter outlines the practical steps to build the LLM component of the project, from setting up the environment and acquiring data to preprocessing, fine-tuning and training the model, with careful attention to the specific formatting for the mistral model[13] and mistral-finetune[19]. It also covers the challenges faced during training and inference[22], and demonstrates how generated SPARQL queries were used with the DBpedia Knowledge Graph. Although the complete KGQA pipeline was not completed due to time, the successful implementation of LLM provides a basis for future evaluation, and the chapter discusses implementation issues such as hardware and data constraints, as well as KGQA complexities, highlighting areas for future work and serving as a practical realization of the project design in the next chapter on evaluations 5.

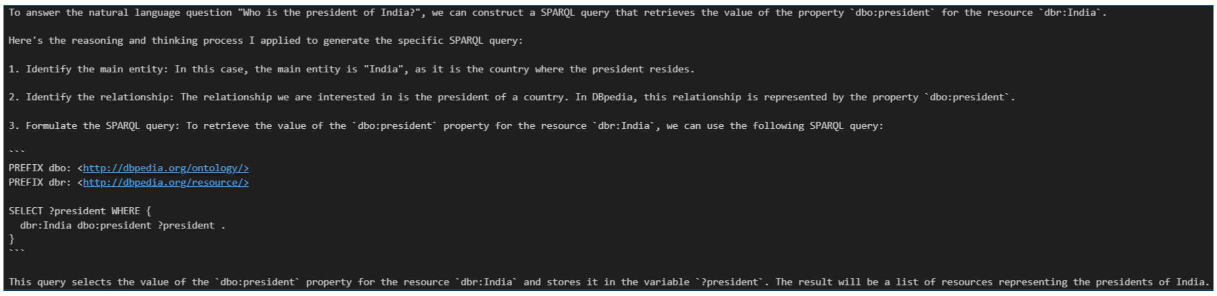
Chapter 5

Evaluation

This chapter discusses the evaluation of the whole project carried out, specifically with respect to the results which are not complete due to the reasons aforementioned. However, there is no empirical evaluation done since no experiments with different scenarios were performed. Rather the LLM part of the project has been evaluated and its outputs analyzed in detail. The evaluation of the LLM outputs with respect to the Natural language question posed and the belief made in this project is as follows: "Who is the president of India?", say is the posed query to the LLM. Since the LLM has been trained for the conversion task, it should give the corresponding SPARQL[5] query, which it gives as shown in figure 5.1. The LLM also provides the reasoning behind generating that specific SPARQL query since it was asked in the system prompt as well. On observation, it was found that the LLM indeed provides the wrong SPARQL query for the question posed. However, it understands the SPARQL syntax well enough such as the URIs[5] for the prefixes but is not good at constructing the query. Although the query would seem correct semantically but is incorrect in its own intricacies. It understands the resources, relations and entities well but not convincingly well. It also understood the question that is being asked. As a consequence, the language output by the LLM with respect to the SPARQL query, not the reasoning, seems good and fluent but is not correct at all. So, it would be wrong to say that the LLMs are good at language completely because the wrong query generation could be highly biased by the dataset-its quality, correctness of the NL-SPARQL queries, syntactic flaws and others. Hence, with respect to the project's consideration, the assessment of the LLM completely being good at language is considerably true but not convincingly. On the other hand, the language outputted for the reasoning behind seems to be good and fluent enough to convince that it is indeed true. In addition, the reasoning is quite convincing for why the LLM did what it did. But this cannot be taken as a strong evidence for the belief to be completely true. However, it performs bad at the syntactic understanding of the SPARQL as a query language and the DBpedia[16] Knowledge Graph structure, in general. There are nuances thus, that are crucial for the LLM to understand and learn for it to be convincingly good at language. Moreover, the outputs need to be verified with the help of KGQA[4, 2] system for better evaluation along with the attention mechanism[17] for the reasoning context understanding to verify. In conclusion, the belief could not be completely stated as being true or false but is somewhere in the middle with some strong observations made mentioned earlier. Also, the language and reasoning on just the output language perspective seem to align[3] but are not exactly. The reasoning given also needs to be more detailed for making any inferences about the output and its evaluation. Therefore, no concrete conclusions can be made yet as needs further work to evaluate.

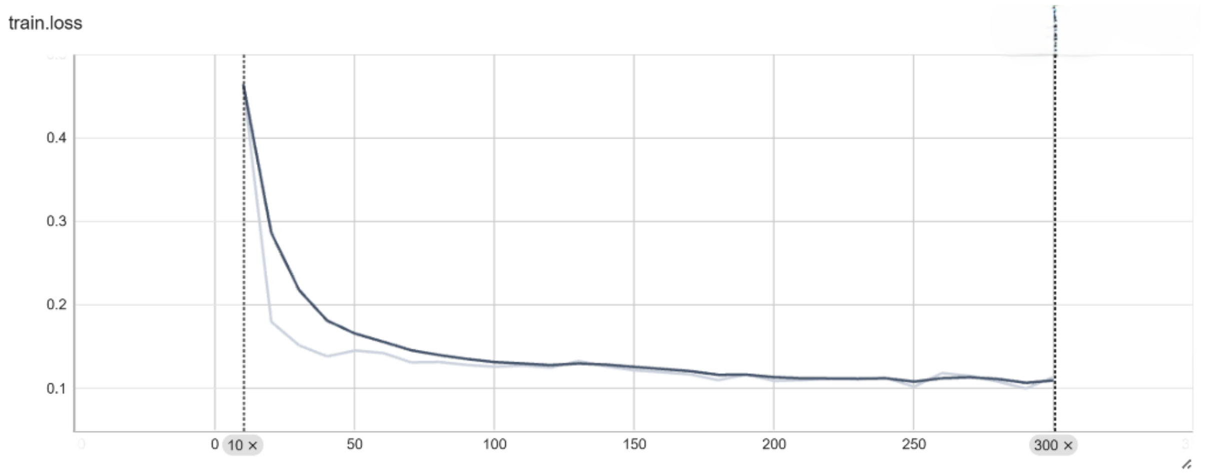
5.1 Results

This section presents all the results whether it was the LLM output or the LLM fine-tuning phase outputs.



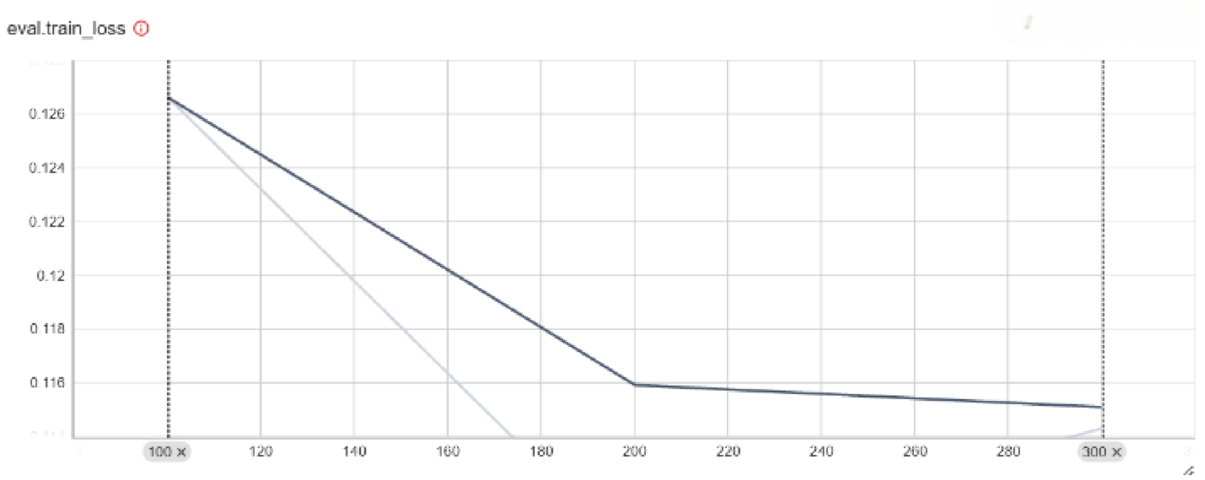
~~~
To answer the natural language question "Who is the president of India?", we can construct a SPARQL query that retrieves the value of the property `dbo:president` for the resource `dbr:India`.

Here's the reasoning and thinking process I applied to generate the specific SPARQL query:

1. Identify the main entity: In this case, the main entity is "India", as it is the country where the president resides.

2. Identify the relationship: The relationship we are interested in is the president of a country. In DBpedia, this relationship is represented by the property `dbo:president`.

3. Formulate the SPARQL query: To retrieve the value of the `dbo:president` property for the resource `dbr:India`, we can use the following SPARQL query:

```
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX dbr: <http://dbpedia.org/resource/>

SELECT ?president WHERE {
  dbr:India dbo:president ?president .
}
```

This query selects the value of the `dbo:president` property for the resource `dbr:India` and stores it in the variable `?president`. The result will be a list of resources representing the presidents of India.
~~~

Figure 5.1: LLM output with generated SPARQL Query for the NL question and the reasoning behind its generation.



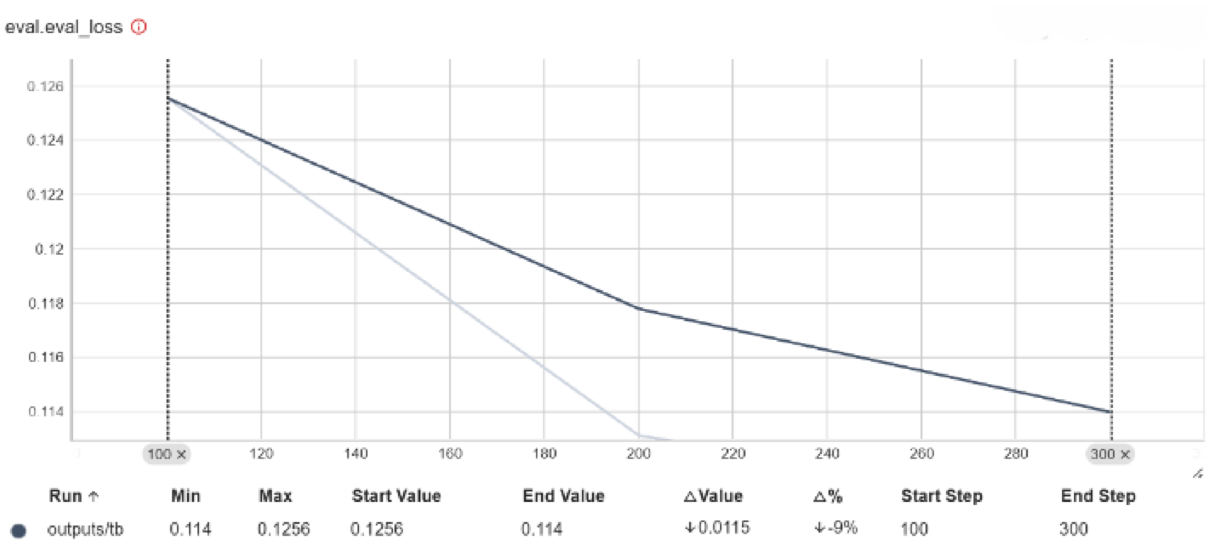
Run ↑	Min	Max	Start Value	End Value	△Value	△%	Start Step	End Step
● outputs/tb	0.1074	0.4644	0.4644	0.1102	↓0.3542	↓-76%	10	300

Figure 5.2: Output plot of the LLM training loss. X-axis represents the number of training steps for LLM and the Y-axis represents the loss values. Similar scale for the subsequent plots.



Run ↑	Min	Max	Start Value	End Value	△Value	△%	Start Step	End Step
● outputs/tb	0.1151	0.1266	0.1266	0.1151	↓0.0115	↓-9%	100	300

Figure 5.3: Output plot of the training loss while evaluation of the LLM. Precisely, training loss evaluated at the evaluation phase.



	Run ↑	Min	Max	Start Value	End Value	∆Value	∆%	Start Step	End Step
●	outputs/tb	0.114	0.1256	0.1256	0.114	↓0.0115	↓-9%	100	300

Figure 5.4: Output plot of the validation loss of the LLM.

5.2 Summary

This chapter details the initial look at how well the project is working, focusing on the LLM since the whole system is not finished.The LLM was asked a question, and it gave a SPARQL query and explained its thinking as in figure 5.1. Although it seemed to grasp the question and the basic SPARQL requirements, the generated answer was actually incorrect. However, the explanation seemed reasonable. It was found that the LLM understands general ideas, but struggles with precise rules of SPARQL and DBpedia KG structure. So, it cannot be definitively said if the belief about the model's language skills and its alignment with reasoning is right yet; the system needs to be completed and more testing to be done to get a clearer picture.

Chapter 6

Conclusions & Future Work

This chapter summarizes the key findings of the project, drawing conclusions from the evaluation results presented in Chapter 5. The alignment between language fluency and reasoning capabilities of the Large Language Model (LLM)[3] is assessed, highlighting its strengths and weaknesses. The chapter also identifies areas for future work that extend the scope of this project, including further testing, improvements to the system, and the integration of additional components for a more comprehensive evaluation.

6.1 Conclusions

The project aimed to explore the alignment between the language outputs of Large Language Models (LLMs) and the underlying reasoning processes behind them[3]. Specifically, it focused on assessing whether the LLM-generated SPARQL[5] queries and their associated reasoning matched the intent of the posed natural language questions. Knowledge Graphs (KG)[1], and more precisely, a KGQA system[4] using DBpedia[16], were intended to serve as verification tools for this alignment. The final system was designed with two main components: a fine-tuned LLM responsible for converting natural language queries into SPARQL and generating corresponding reasoning, and a KGQA system to act as a reference for correctness. Due to time constraints, the full integration of KGQA could not be completed and the evaluation focused solely on the LLM component. From the results obtained, it was observed that the LLM was able to parse and interpret natural language questions with reasonable fluency and provided explanations that were grammatically coherent and appeared logically sound. However, the generated SPARQL queries often lacked syntactic correctness or accurate alignment with the DBpedia structure. This indicates that although the LLM captured the intent of the questions to some extent, it did not consistently construct correct or executable[8] SPARQL queries. The reasoning outputs were notably more fluent and persuasive than the queries themselves, but upon closer inspection, did not always reflect the actual logic used in the SPARQL query generation. This misalignment points to a gap between language fluency and reasoning accuracy. Therefore, the belief that LLMs are strong in language but weaker in structured reasoning holds partial truth in the context of this project. In summary, while the LLM demonstrated promising abilities in understanding and generating language, the more in-depth evaluation showed that reasoning, especially in structured tasks such as SPARQL generation, remains a challenge. The results support the idea that fluency in the language does not necessarily equate to reliable reasoning. This project establishes a foundation for future investigations of the language-thought alignment of LLMs[3], using knowledge-grounded evaluation frameworks like KGQA[1][2][4].

6.2 Future Work

There is a lot of future work possible in the respective field of the project. This can also be understood by going through the complete report. However, one of the foremost future works is to complete the implementation of the designed approach for the problem formulated in the project. The LLM part has been completed. The KGQA part has to be implemented from scratch as well as the attention-employed mechanism[17] to understand the reasoning context given by the LLM itself to verify the alignment claim[3] and similar to that discussed in the previous chapters. However, along with the completion of the project itself in implementation and thus the evaluations, the interesting questions still to ask are "Can LLMs really think ?", "Is the approach designed in the project even viable ?", "If viable then is there any concrete evidence for the alignment ?" and there should be a quest still going on along these lines only until strong evidence is found. This is because these types of question have been studied for the Human Brain from which the very inspiration of AI, in general, comes. Another possible area of work is to make LLMs as hallucination-free as possible and bring them closer to their very inspiration as possible in the way it answers as the brain answers. In simpler terms, the Human Brain may come up with its own made-up information in form of "opinions" but still it does not dwell on them and hence does not always mention them with the most important capability of them changing unless formed into perceptions. On the other hand, LLM talking come with made up information and dwell on them until told that they are wrong. So, they should behave very much like the human brain in conversations, not much more than that. They should be transparent in what they say or do. The scope of further work to get these LLMs to understand complex or multi-hop questions to generate the corresponding SPARQL[5] queries is huge. This is because of the KG structure, inability to understand the natural language question completely and similar. This question is not only limited to LLMs, but these KGQA[2][4] systems also face this issue along with many other their own sets of challenges. The last area for further work and future research is the use of LLMs in integration with KG[1] to perform reasoning tasks for question answering[18]. This work may show a path into more depth of these AI systems and hence, probably the path towards AGI. Reasoning is a very authentic capability of the Human Brain, one should remember. In conclusion, the aforementioned questions are crucial to be answered before moving forward into deep further work since these will act as a bridge between the prior research done so far and the further research to come taking into account most importantly the cognitive sciences as well.

Bibliography

[1] B. Xu, “Knowledge graph learning,” https://github.com/BrambleXu/knowledge-graph-learning, 2020, accessed: 2025-04-19.

[2] H. Sherry, “Knowledge graph question answering (kgqa),” https://github.com/heathersherry/Knowledge-Graph-Tutorials-and-Papers/blob/master/topics/Knowledge%20Graph%20Question%20Answering%20(KGQA).md, 2021, accessed: 2025-04-19.

[3] K. Mahowald, A. A. Ivanova, I. A. Blank, N. Kanwisher, J. B. Tenenbaum, and E. Fedorenko, “Dissociating language and thought in large language models,” *Trends in Cognitive Sciences*, vol. 28, no. 6, pp. 517–540, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1364661324000275

[4] D. Diefenbach, V. Lopez, K. Singh, and P. Maret, “Core techniques of question answering systems over knowledge bases: a survey,” *Core Techniques of Question Answering Systems over Knowledge Bases: a Survey*, vol. 55, 06 2018.

[5] W3C, “Sparql 1.1 query language,” https://www.w3.org/TR/sparql11-query/, 2013, w3C Recommendation, Accessed: 2025-04-19.

[6] H. Han, Y. Wang, H. Shomer, K. Guo, J. Ding, Y. Lei, M. Halappanavar, R. A. Rossi, S. Mukherjee, X. Tang, Q. He, Z. Hua, B. Long, T. Zhao, N. Shah, A. Javari, Y. Xia, and J. Tang, “Retrieval-augmented generation with graphs (graphrag),” 2025. [Online]. Available: https://arxiv.org/abs/2501.00309

[7] Y. Gao, Y. Xiong, X. Gao, K. Jia, J. Pan, Y. Bi, Y. Dai, J. Sun, M. Wang, and H. Wang, “Retrieval-augmented generation for large language models: A survey,” 2024. [Online]. Available: https://arxiv.org/abs/2312.10997

[8] DBpedia Association, “Dbpedia sparql endpoint,” https://dbpedia.org/sparql, 2025, accessed: 2025-04-19.

[9] W3C, “Rdf 1.2 concepts and abstract syntax,” https://www.w3.org/TR/rdf12-concepts/, 2024, w3C Candidate Recommendation, Accessed: 2025-04-19.

[10] W. Zheng, J. X. Yu, L. Zou, and H. Cheng, “Question answering over knowledge graphs: Question understanding via template decomposition,” *Proc. VLDB Endow.*, vol. 11, pp. 1373–1386, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:51995949

[11] T. A. Taffa and R. Usbeck, “Leveraging llms in scholarly knowledge graph question answering,” 2023. [Online]. Available: https://arxiv.org/abs/2311.09841

[12] Unsloth Team, “Unsloth documentation,” https://docs.unsloth.ai/, 2024, accessed: 2025-04-19.

[13] Mistral AI, “Mistral-7b-instruct-v0.3,” https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.3, 2024, accessed: 2025-04-19.

[14] Ollama, “Ollama: Run llms locally,” https://ollama.com/, 2024, accessed: 2025-04-19.

[15] L. Xu, H. Xie, S.-Z. J. Qin, X. Tao, and F. L. Wang, “Parameter-efficient fine-tuning methods for pretrained language models: A critical review and assessment,” 2023. [Online]. Available: https://arxiv.org/abs/2312.12148

[16] DBpedia Association, “Dbpedia: A crowd-sourced community effort to extract structured information from wikipedia,” https://www.dbpedia.org/, 2025, accessed: 2025-04-19.

[17] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention is all you need,” 2023. [Online]. Available: https://arxiv.org/abs/1706.03762

[18] M. Yasunaga and J. Leskovec, “Qagnn: Reasoning with language models and knowledge graphs for question answering,” https://snap.stanford.edu/qagnn/, 2021, accessed: 2025-04-19.

[19] Mistral AI, “mistral-finetune: Fine-tuning framework for mistral models,” https://github.com/mistralai/mistral-finetune, 2024, accessed: 2025-04-19.

[20] J. César-P, “Question-sparql dataset,” https://huggingface.co/datasets/julioc-p/Question-Sparql, 2023, accessed: 2025-04-19.

[21] Vast.ai, "Vast.ai: Simplified cloud computing for ai," https://vast.ai/, 2025, accessed: 2025-04-19.

[22] Mistral AI, "mistral-inference: Inference library for mistral models," https://github.com/mistralai/mistral-inference, 2024, accessed: 2025-04-19.

[23] Mistral AI Inference, "mistral-inference github issues," https://github.com/mistralai/mistral-inference/issues, 2024, accessed: 2025-04-19.