Bachelor of Computer Science
may 2025

Benchmarking Approaches for QnA LLMs with SPARQL on EnterpriseData

Integrating RAG, ReAct, Multiagent for query generation with Knowledge Graphs

Morris Simons

Faculty of Faculty, Blekinge Institute of Technology, 371 79 Karlskrona, Sweden

This thesis is submitted to the Faculty of Faculty at Blekinge Institute of Technology in partial fulfillment of the requirements for the degree of Bachelor of Computer Science. The thesis is equivalent to Weeks weeks of full-time studies.

The authors declare that they are the sole authors of this thesis and that they have not used any sources other than those listed in the bibliography and identified as references. They further declare that they have not submitted this thesis at any other institution to obtain a degree.

Contact Information:
Author:
Morris Simons
E-mail: mosi21@student.bth.se

Co-advisor
Name & title: Katariina Kari, Head of Data at in-parallel
E-Mail: katsi@in-parallel.com
Compnay: In-parallel

University advisor:
Dr. Prashant Goswami
Title Firstname Lastname
Department of Department of Computer Science (DIDA) Department

Faculty of Faculty
Blekinge Institute of Technology
SE–371 79 Karlskrona, Sweden

Internet : www.bth.se
Phone : +46 455 38 50 00
Fax : +46 455 38 50 57

Abstract

Background: Large Language Models (LLMs) offer a promising avenue for Natural Language Question Answering (NLQA) on KGs, enabling users to "chat with their data" by converting natural language to SPARQL queries. However, effectively applying LLMs in complex enterprise environments with proprietary KGs presents challenges, particularly regarding context management and the lack of comparative evaluations of different advanced methodologies.

Objectives: This thesis aims to systematically evaluate and compare Text-to-SPARQL generation methodologies Direct LLM Prompting, Retrieval-Augmented Generation (RAG), Reasoning and Action (ReAct) frameworks, Multi-Agent Architectures, and a pre-built Langchain integration applied to an enterprise KG. The primary objective is to assess their performance regarding query accuracy, response latency, token usage, and robustness, providing empirically-grounded insights into their practical suitability for enterprise data.

Methods: A quantitative, experimental methodology was employed using the GPT4.1-mini-2025-04-14 LLM. Five distinct Text-to-SPARQL approaches were implemented and benchmarked against a proprietary enterprise KG using 30 curated question-query pairs, with 7 runs per question. Performance was measured by Execution Accuracy (EA), Overall Execution Accuracy (OEA), Average OEA (AOEA), response latency (Overall Average Latency OAL), token usage (Overall Average Total Tokens - OATT), and query execution success rate (robustness).

Results: The RAG approach achieved the highest accuracy (0.424 AOEA) and lowest latency (3.1s OAL). The Multi-Agent system was the most token-efficient (1689 average total tokens) and demonstrated high robustness (95% success rate). Direct Prompting was the most token-expensive (22668 average total tokens) with moderate accuracy (0.343 AOEA). The ReAct agent, in its current configuration, generally underperformed across multiple metrics. No method excelled universally.

Conclusions: The study concludes that the optimal Text-to-SPARQL method for enterprise KGs depends on specific operational priorities. RAG offers a strong balance of accuracy and speed. Multi-Agent systems are promising for cost-efficiency and robustness. Pre-built integrations can offer high execution reliability but may sacrifice accuracy and cost-effectiveness on specific KGs. The findings provide empirical evidence on the trade-offs of these LLM-based techniques in a realistic, data-scarce enterprise context.

Keywords: Chat with your data, Natural Language Querying, Text-to-SPARQL, Knowledge Graphs, GraphRAG, Large Language Models

Acknowledgments

I also extend my sincere thanks to In-Parallel for providing the data that made this research possible and to Katariina Kari for her insightful guidance and mentorship on knowledge graphs, which were crucial in shaping the direction of this work.

Contents

Abstract i

Acknowledgments iii

1 Introduction 3
- 1.1 Problem statement . . . 4
- 1.2 Aim and Objectives . . . 6
- 1.3 Thesis Outline . . . 7

2 Background 9
- 2.1 Core concepts of SPARQL . . . 9
- 2.2 Ethical, Societal, and Sustainability Aspects . . . 10

3 Related Work 13
- 3.1 History . . . 13
- 3.2 Leveraging Text-to-SQL Advancements for SPARQL Generation . . . 13
- 3.3 The Challenge of Fine-tuning for Enterprise Text-to-SPARQL . . . 15
- 3.4 The Importance of Enterprise Datasets . . . 16
- 3.5 Research GAP and summary . . . 17

4 Method 19
- 4.1 Research question . . . 19
- 4.2 Research Approach and Strategy . . . 19
- 4.3 Methodology . . . 20
- 4.4 Literature Review Process Overview . . . 20
  - 4.4.1 Search Scope and Source Diversification . . . 20
- 4.5 Environment Setup . . . 20
- 4.6 Model settings and prompt details . . . 20
- 4.7 Key Python Libraries and Frameworks . . . 22
  - 4.7.1 Experiment 1: Direct LLM Prompting (Baseline) . . . 25
  - 4.7.2 Experiment 2: RAG-Based Approach . . . 26
  - 4.7.3 Experiment 3: ReAct-Based Agent . . . 27
  - 4.7.4 Experiment 4: Multi-Agent System . . . 28
  - 4.7.5 Exp 5 Ontotext GraphDB QA LangChain intergration . . . 30
- 4.8 Evaluation . . . 31
  - 4.8.1 Accuracy . . . 32
  - 4.8.2 Latency . . . 33
  - 4.8.3 Cost . . . 33

4.8.4 Robustness 33

5 Results and Analysis **35**
5.1 Results 35
5.1.1 Execution Accuracy 36
5.1.2 Latency 37
5.1.3 Token Usage 37
5.1.4 Robustness 37
5.1.5 Comprehensive Statistical Summary (Std 95% CI) 42
5.2 Summary Metrics 42
5.3 Analysis 43
5.3.1 Execution Accuracy 43
5.3.2 Latency 43
5.3.3 Token Usage 43
5.3.4 Robustness (Success Rate) 44
5.4 Summary of Results 44
5.5 Summary Metrics 45
5.6 Rank Summary Metrics 45

6 Discussion **47**
6.1 Addressing the Research Question 47
6.1.1 Comparative Performance Analysis Summary 47
6.1.2 Synthesis: Answering the Research Question 48
6.1.3 The Ascendancy of RAG: Focused Context vs. Comprehensive Knowledge 49
6.1.4 The Cost of Agency: ReAct and Multi Agent Systems 49
6.1.5 Pre-built Integrations: Robustness vs. Specificity 49
6.1.6 Unexpected Findings and Conflicting Data 50
6.2 Reflections on Methodological Choices and Observations 50
6.2.1 Prompting Strategies 50
6.2.2 Influence of LLM Model 51
6.2.3 The TOP-K Parameter 51
6.2.4 Execution Accuracy (EA) 51
6.3 Limitations of the Study 53
6.4 Summary and Position of this Thesis 53
6.5 Contribution to Closing Research Gaps 54

7 Conclusions and Future Work **55**
7.1 Conclusions 55
7.2 Future Work 56

References **59**

A Supplemental Information **65**
A.1 Summary Metrics on question 1 66
A.2 Summary Metrics 66
A.3 Experimental Prompts 66

Chapter 1

Introduction

The ability to effectively access and derive actionable insights from vast and complex datasets is a cornerstone of modern enterprises. However, traditional methods of data interaction often require users to possess specialized technical skills, such as proficiency in query languages like SQL or the SPARQL Protocol and RDF Query Language (SPARQL), thereby limiting direct data access for a broader audience. Natural Language Processing (NLP), a field within Artificial Intelligence (AI), is dedicated to enabling computers to understand, interpret, process, and generate human language. A key objective within NLP has long been the development of intuitive interfaces that allow users to interact with data using natural language, mitigating the need for formal query expertise. NLP encompasses diverse applications, including machine translation, sentiment analysis, text summarization, and, central to this research, Natural Language Question Answering (QA) systems. Another common exact term for this context is Natural Language Querying (NLQ).

The advent of Large Language Models (LLMs), such as OpenAI's GPT series [2, 26, 28], represents a paradigm shift in NLP. These models, pre-trained on extensive text corpora, have demonstrated remarkable capabilities in understanding context, generating fluent text, and performing various reasoning tasks. Consequently, LLMs have significantly impacted natural language processing applications, enabling more efficient text generation, summarization, dialogue systems, and notably, enhancing the potential of QA systems.

In a blog post from 2012 Amit Singhal at Google [34] wrote an article called: Introducing the Knowledge Graph: things, not strings. Where he outlined The Knowledge Graph (KGs) enhances Google Search in three main ways to start:

- **Find the right thing:** Language can be ambiguous—do you mean Taj Mahal the monument, or Taj Mahal the musician? The results were more relevant because they understood these entities, and the nuances in their meaning, the way you do.

- **Get the best summary:** With the Knowledge Graph, Google could better understand your query, so they can summarize relevant content around that topic, including key facts you're likely to need for that particular thing. For example, if you're looking for Marie Curie, you'll see when she was born and died, but you'll also get details on her education and scientific discoveries: For example, people are interested in knowing what books Charles Dickens wrote,

whereas they're less interested in what books Frank Lloyd Wright wrote, and more in what buildings he designed. It also models all these inter-relationships. It's the intelligence between these different entities that's the key.

- **Go deeper and broader:** Finally, the part that's the most interesting of all—the Knowledge Graph can help you make some unexpected discoveries. You might learn a new fact or new connection that prompts a whole new line of inquiry.

With the rise of LLMs, their application to Natural Language Question Answering (QA) has become particularly promising for democratizing data access. This enables users to query diverse data sources, including complex enterprise Knowledge Graphs, using conversational questions rather than specialized query languages. This paradigm, often referred to as *Chat with your data*, aims to make data interaction significantly more intuitive and accessible. While the ambition of querying data via natural language is not new (the history of which is explored further in Chapter 3), the advanced capabilities of modern LLMs have revitalized this area, offering new avenues for creating sophisticated Text-to-SPARQL systems for enterprise KGs. However, effectively harnessing LLMs for this purpose in complex enterprise environments introduces a distinct set of specific challenges that this thesis aims to investigate.

1.1 Problem statement

Pre-trained neural language models have been shown to learn a substantial amount of in-depth knowledge from data [27]. They can do so without any access to an external memory, as a parameterized implicit knowledge base [29, 31]. But this parameter memory sometime needs additional external information. This has given the rise of Retrieval-Augmented Generation (RAG) to give the LLM additional context for a problem. For more on the topic of RAG you can read Patrick Lewis 2020 paper on RAGs [20]

LLMs operate by predicting subsequent tokens based on provided context, making context management essential [45]. However, improper context handling can either confuse the model or lead it to misinterpret instructions. The effect of too much context can be seen from lost in the middle paper [22] and the effect of to little context gives the model too few instructions often creates ambiguity in more complex tasks as explained in many prompting guides such as Anthropic's Prompt Engineering Interactive Tutorial [1]:

- Too little context → Ambiguity and misunderstanding
- Too much context → Model confusion or memory limitations

There are multiple ways to store data for RAGs to give as context for the LLM and a common way has become to store data into vectors. According to Yikun

Han, Chunjiang Liu, and Pengfei Wang in the paper A Comprehensive Survey on Vector Database [11] The vectors are usually generated by applying some kind of transformation or embedding function to the raw data, such as text, images, audio, video, and others. The embedding function can be based on various methods, such as machine learning models, word embeddings, and feature extraction algorithms. Each vector can be mapped out into a vector space for us to search in this vector space. Vector databases have several advantages over traditional databases.

- Fast and accurate similarity search and retrieval.
- Support for complex and unstructured data

But several disadvantages as noted in this article here are some of them [41].

- Lack of structured data management can create complexity.
- understanding of relational meaning

In the same way that Google [34] discovered that in order to master search, they needed to go beyond mere textual analysis and map out the underlying entities and relationships in new things underneath the strings, we are beginning to see the same pattern emerge in the world of AI. This have given rise to the concept of Graph RAG [5]

> **things, not strings:** *As the fruit apple is not the same as the company Apple but they share the same string and might share a similar vector space. [30]*

Despite recent technological advancements, a significant gap persists in the systematic benchmarking and comparative analysis of different LLM-based approaches for Text-to-SPARQL generation, particularly within realistic enterprise settings. Existing research often concentrates on public datasets or fails to comprehensively evaluate the multifaceted trade-offs (including accuracy, cost, and latency) associated with various sophisticated interaction patterns such as direct prompting, RAG, ReAct, Langchain integration or multi-agent systems.

This evaluation is especially critical when addressing proprietary KGs and the common enterprise scenario of data scarcity, which frequently precludes extensive model fine-tuning. Consequently, enterprises currently lack clear, empirically backed guidance on selecting and implementing strategies that offer optimal performance and practical viability, a factor that can potentially hinder the effective adoption and tangible impact of these powerful technologies in real-world business intelligence and data interaction scenarios.

Therefore, the central problem this thesis addresses is the lack of robust, comparative evaluation to determine the most effective and efficient LLM-based methodologies for generating accurate SPARQL queries from natural language questions within complex enterprise knowledge graph environments, considering the practical constraints and performance requirements of such settings. This research aims to fill this void by providing empirical evidence on the performance of several distinct contemporary approaches.

1.2 Aim and Objectives

The primary aim of this thesis is to systematically evaluate and compare distinct contemporary Text-to-SPARQL generation methodologies applied to enterprise knowledge graphs. This comparative analysis will focus on assessing their performance in terms of accuracy, response latency, and robustness, thereby providing empirically-grounded insights into their practical suitability for enterprise applications.

To achieve this aim, the following sequential and measurable objectives will be pursued:

1. **Baseline System Implementation:** Develop and document a foundational Text-to-SPARQL system utilizing direct prompting of Large Language Models (LLMs). This system will serve as the performance baseline against which more advanced methodologies are compared.

2. **Benchmarking Framework Establishment:** Design and implement a comprehensive benchmarking framework specifically tailored for evaluating Text-to-SPARQL systems within an enterprise context. This framework will establish key performance metrics, including query accuracy, response latency, and robustness.

3. **Comparative Evaluation of Methodologies:** Conduct a rigorous comparative analysis of the following Text-to-SPARQL generation approaches using the established benchmarking framework:

   - Retrieval-Augmented Generation (RAG)
   - Reasoning and Action (ReAct) framework
   - Multi-Agent Architecture
   - Ontotext GraphDB QA LangChain intergration

   This objective involves executing experiments for each methodology, measuring its performance against the defined metrics, and analyzing the results to delineate their respective strengths and weaknesses in enterprise knowledge graph scenarios.

These methodologies represent a spectrum of strategies, from direct LLM application to more intricate retrieval, reasoning, and collaborative agent-based techniques. Their systematic comparison is intended to elucidate their capabilities and limitations in real-world enterprise settings.

The central research question (RQ) guiding this investigation is:

> **RQ:** *How do different advanced Text-to-SPARQL methods (Direct Prompting, RAG, ReAct and Multi-Agent Architectures) compare regarding accuracy, cost, response latency, and robustness within enterprise knowledge graph scenarios?*

To answer this research question, a benchmark-driven evaluation employing enterprise-grade data and representative query scenarios will be conducted. This approach ensures that findings are grounded in realistic conditions, offering impactful insights for industry practitioners. The evaluation will quantitatively measure key performance indicators such as execution accuracy, response latency, computational cost (e.g., token consumption), and robustness. Furthermore, the study will include a qualitative analysis of development effort and discuss the privacy implications and scalability potential inherent to each evaluated methodology.

1.3 Thesis Outline

The remainder of this thesis is structured to systematically address the research question and present the investigation's findings, organized as follows:

- **Chapter 1: Introduction** lays the groundwork for the thesis. It introduces the domain of Text-to-SPARQL for enterprise knowledge graphs, articulates the problem statement concerning the need for comparative benchmarking of LLM-based methodologies, and defines the aim, objectives, and the central research question guiding this study. This chapter concludes with the present outline of the thesis structure.

- **Chapter 2: Background** provides foundational knowledge necessary to understand the context of this research. It covers the core concepts of SPARQL and its role in querying knowledge graphs. This chapter also addresses the ethical, societal, and sustainability aspects relevant to the research, particularly concerning the use of enterprise data and AI technologies.

- **Chapter 3: Related Work** offers a comprehensive review of existing literature. It delves into the history of natural language interfaces, discusses advancements in Text-to-SQL that inform Text-to-SPARQL generation, examines the challenges and data requirements for fine-tuning LLMs in enterprise settings, and underscores the importance of using enterprise-specific datasets for evaluation. The chapter culminates in identifying the specific research gaps this thesis aims to address.

- **Chapter 4: Method** details the research design and experimental methodology. It restates the research question and describes the overall research approach and strategy. This includes an overview of the literature review process, the model settings and prompt details for the LLM, the specific implementation details for each of the five evaluated Text-to-SPARQL approaches (Direct Prompting, RAG, ReAct, Multi-Agent system, and Ontotext GraphDB QA LangChain integration), and the precise metrics (Accuracy, Latency, Cost, Robustness) and procedures used for evaluation.

- **Chapter 5: Results and Analysis** presents the empirical data gathered from the benchmark experiments. The performance of each Text-to-SPARQL method is reported quantitatively across the defined metrics, supported by tables and figures. This is followed by a detailed analysis of these results,

comparing the approaches based on execution accuracy, latency, token usage, and robustness. The chapter concludes with a summary of the key findings from the results.

- **Chapter 6: Discussion** interprets and contextualizes the findings presented in the Results and Analysis chapter. It addresses the research question by synthesizing the comparative performance analysis, discusses the ascendancy of RAG, the cost and complexities of agentic systems, and the trade-offs of pre-built integrations. The chapter also reflects on methodological choices, unexpected findings, the limitations of the study, summarizes the thesis's position, and highlights its contribution to closing identified research gaps.

- **Chapter 7: Conclusions and Future Work** summarizes the main conclusions drawn from the research, reiterating the answers to the research question and the key contributions of the thesis. It also candidly discusses the limitations of the study and proposes several promising directions for future research to further advance Text-to-SPARQL capabilities for enterprise knowledge graphs.

Chapter 2

Background

2.1 Core concepts of SPARQL

As SPARQL plays a central role in this work, a brief overview is provided here. We begin with RDF (Resource Description Framework) [24], a standardized model developed by the W3C (World Wide Web Consortium) for describing web resources and metadata. RDF represents data as a collection of triples:

Subject → Predicate → Object

Each triple expresses a relationship or property of a resource. An example is SPARQL [25] (pronounced "sparkle"), which stands for SPARQL Protocol and RDF Query Language. SPARQL is the standard query language for retrieving, manipulating, and querying data stored in RDF format. It is analogous to SQL but specifically designed for graph-structured data.

A simple example of how an RDF looks like with real data.

```
:John_Lennon → :member → :The_Beatles
```

Figure 2.1 shows a more complex example. With multiple RDFs, in this structure of entities such as songs, writers, albums, and release dates are interconnected into a rich graph of relationships. This web of complex relationships is called a knowledge graph when querying the Knowledge Graph (KG), we use SPARQL. 2.1 also illustrates how SQL data can be transformed into a Knowledge Graph. An example query is shown in listing 2.1

Listing 2.1: SPARQL Query to retrieve all members of The Beatles

```
PREFIX ex: <http://example.org/property/>

SELECT ?person
WHERE {
  ?person ex:member ex:The_Beatles.
}
```

The following SPARQL query in 2.1 retrieves all members associated with the band The Beatles. In this work, the goal is to automatically generate such SPARQL queries using Large Language Models (LLMs).

The mappings and their "rules of relations" aren't automatically generated; they're

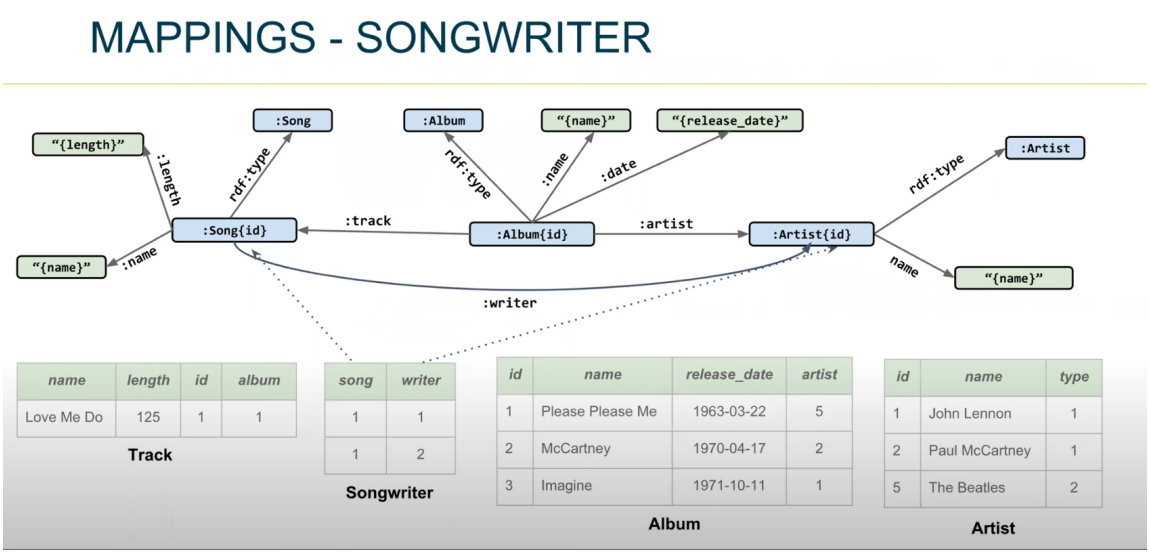

Figure 2.1: An SQL database extended with a Knowledge Graph layer.

explicitly defined by the mapping author, based on their intended semantic meaning. Creating rules for relations and mappings between data and semantic representations is a fundamental responsibility of an ontologist.

2.2 Ethical, Societal, and Sustainability Aspects

The use of real-world enterprise data is core to this thesis, aiming for a realistic evaluation of Text-to-SPARQL methods (RQ). This approach enhances the fairness of the benchmark itself, reflecting actual operational complexities.

For organizations it is important that the results that model gives to the user is as accurate as possible this is why this kind of research is important. As inaccurate information can cause decision makers to make wrong decision or decisions on inaccurate information. It is also important to educate a display what type of accuracy is also possible to give light that the models in it current sate is not accurate enough for many common Businesses operation questions. This research will directly through our partnership with in-parallel.

However, it necessitates acknowledging potential inherent biases within the enterprise data and the foundational LLM (GPT4.1-mini-2025-04-14) [10]. Care must be taken to ensure the system does not inadvertently perpetuate skewed representations.

Protecting the confidentiality of In-Parallel's proprietary data has been a paramount concern throughout this study. The data, by its nature, is sensitive and commercially valuable. To ensure its protection, several measures were implemented:

- A Non-Disclosure Agreement (NDA) was established with In-Parallel, legally binding the research to strict confidentiality terms.

- Data access and experimentation were conducted within a secure environment specified by In-Parallel, adhering to their data handling protocols.
- For this thesis, specific examples of data instances, queries, and knowledge graph schema details have been generalized or anonymized. For instance, the 30 question-query pairs used in the benchmark are presented with anonymized labels to protect proprietary information, as detailed in Chapter 5.

These necessary protective measures inevitably impact the transparency and external reproducibility of this research. The full dataset and knowledge graph schema cannot be publicly released, which limits the ability of other researchers to directly replicate the experiments. This limitation is acknowledged; however, the use of genuine enterprise data provides unique insights into real-world challenges and performance that are often unattainable with public datasets. The methodologies, agent designs, prompt structures, and the evaluation framework are documented as clearly as possible within these constraints to offer maximum possible transparency.

From a sustainability perspective, the reliance on powerful LLMs and potentially complex agent interactions carries a significant environmental cost due to high energy consumption [16]. This research therefore emphasizes evaluating computational efficiency (e.g., latency, token usage) alongside accuracy, treating efficiency not just as a performance metric but as a vital step towards more sustainable AI solutions. Comparing the resource demands of different approaches (Direct Prompting, RAG, ReAct, Multi-Agent) will offer insights into these trade-offs.

This research aims to contribute positively to the broader technological landscape and society in several ways. By evaluating methods to make complex enterprise knowledge graphs accessible via natural language, this work can help democratize data access for a wider range of users within organizations, potentially reducing reliance on specialized data science teams for routine queries. This can lead to more agile and informed decision-making processes within businesses.

Finally, a commitment to accountable research practices is essential. Robust evaluation using clearly defined metrics like Execution Accuracy (as detailed in Section 4.8) and transparent reporting of system limitations, including potential failure modes, are crucial steps towards this. Understanding when and why the evaluated systems might not perform as expected is vital for responsible interpretation of the results and for guiding future research and potential deployment in any enterprise setting.

Chapter 3

Related Work

3.1 History

The challenge of translating natural language questions into formal database queries has been extensively studied for a longtime, particularly in the context of Text-to-SQL for relational databases.

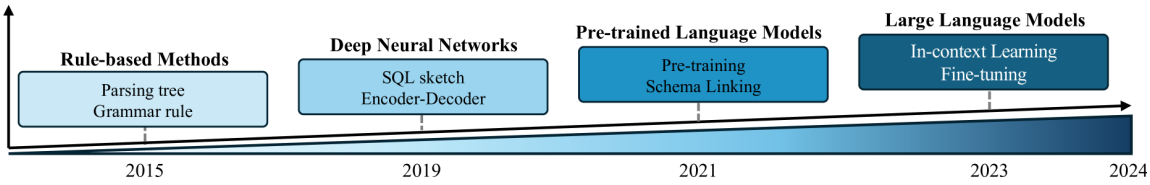

Figure 3.1: Evolutionary process of text-to-SQL research by implementation paradigm. Each stage shows implementation techniques and key representative works. Timestamps are approximate, based on release dates of representative works, with a margin of about one year. Adapted from Hong [14]

Question answering, the ability to interact with data using natural language questions and obtaining accurate results, has been a long-standing challenge in computer science dating back to the 1960s [9, 13, 37]. The field has advanced throughout the past decades [4, 36, 42] through Text-to-SQL approaches, as a means of facilitating chatting with the data that is stored in SQL databases [7, 17, 21, 38, 44].

Relation to this thesis: While these works focus on SQL and relational databases, the fundamental challenge of semantic parsing and mapping natural language to a formal query language is shared. The evolution depicted in Figure 3.1 and the techniques explored in these seminal Text-to-SQL papers provide valuable context and highlight methodologies (like sequence-to-sequence models [17,44]) that have influenced or can be adapted for the Text-to-SPARQL task central to this thesis.

3.2 Leveraging Text-to-SQL Advancements for SPARQL Generation

This chapter reviews existing literature relevant to generating SPARQL queries from natural language, explicitly drawing parallels with established Text-to-SQL

approaches where applicable. We examine several key categories of methods, largely informed by recent research like hong al.'s [15]

Recent foundational surveys, such as the work by Hong. on next-generation database interfaces [15], offer a comprehensive overview of techniques leveraging Large Language Models (LLMs). These techniques have significantly advanced the state-of-the-art in Text-to-SQL performance.

Although this thesis concentrates specifically on the Text-to-SPARQL problem for querying KGs, the methodologies successfully applied in the Text-to-SQL domain serve as a critical source of inspiration and a starting point for our investigation. The core contribution of this thesis involves adapting, applying, and evaluating these LLM-based techniques, initially proven in the SQL world, to address the specific nuances of SPARQL syntax and the graph-based structure of KGs within an enterprise context. Hongs survey [15] is particularly relevant as it categorizes modern LLM-based approaches, providing a framework for the methods explored here.

- **Direct LLM Prompting:** Utilizing LLMs with carefully crafted prompts that include schema information (ontology snippets, relevant triples) to directly generate SPARQL queries.

  Relevance: This serves as a baseline approach against which more sophisticated methods explored in this thesis, such as RAG and agentic frameworks, are compared.

- **Vector Retrieval-Augmented Generation (RAG):** Employing retrieval mechanisms, involving a vector store, to fetch relevant KG snippets (e.g., relevant entities, properties, class information) or similar query examples to augment the context provided to the LLM during generation.

  Relevance: RAG is investigated in this thesis as a method to improve context-awareness and grounding of the LLM's generation process on the specific enterprise KG.

- **Agent-based Systems:** Implementing multi-step reasoning and tool-use frameworks where LLM-powered agents can decompose the problem, interact with the knowledge graph (e.g., lookup entities, validate properties), verify intermediate results, and iteratively construct the final SPARQL query.

  Relevance: This category is central to the methodology explored in this thesis. Inspired by systems like Google's Open Data QnA [8] (mentioned in [15]), we developed a custom agentic solution tailored for Text-to-SPARQL, detailed further in the Methodology chapter.

- **Customized and Fine-tuned Models:** Exploring the use of LLMs specifically fine-tuned on SPARQL generation tasks or domain-specific KG schemas.

  Relevance: Fine-tuning represents a potentially high-performing alternative

approach. We analyze its feasibility and requirements in the next section to justify why it was not pursued in this thesis due to data constraints specific to our enterprise context.

Instead of fine-tuning, this thesis places significant emphasis on exploring more advanced agentic frameworks. Specifically, we investigate the application of the ReAct (Reasoning and Action) framework [39]. ReAct integrates LLM-based reasoning with discrete action steps (like querying the KG or refining the query), offering a structured approach for dynamic, iterative query construction.

Relevance: ReAct represents a concrete implementation strategy within the broader category of agent-based systems and is a core technique evaluated in this work for its potential to handle complex queries and interact effectively with the enterprise KG.

3.3 The Challenge of Fine-tuning for Enterprise Text-to-SPARQL

A prominent approach for adapting LLMs to specific tasks is fine-tuning. However, its feasibility heavily depends on the availability of suitable training data. As noted by Hong. [15] for Text-to-SQL, fine-tuning potentially offers the highest performance ceiling but typically requires substantial task-specific training data (e.g., pairs of natural language questions and corresponding gold standard queries).

This section explains why fine-tuning, despite its potential, was deemed infeasible and therefore not explored experimentally in this thesis. The primary reason is the limited number of curated text-to-SPARQL examples (specifically, 30 pairs) available for the proprietary enterprise KG used in this study. Creating a large dataset of high-quality question-SPARQL pairs requires significant manual effort and domain expertise, which was prohibitive within the scope of this work. This data scarcity contrasts with the requirements suggested by existing literature on fine-tuning for SPARQL generation.

To illustrate the data requirements, consider the following studies:

- **LoRA on Small SPARQL Datasets:** Mecharnia & d'Aquin (2025) [23] successfully fine-tuned LLaMA-based models on the QALD benchmarks (around 400 training pairs) using LoRA (Low-Rank Adaptation). They employed specific regularization techniques (LoRA rank 16, dropout 0.05) to mitigate overfitting on this small dataset, achieving respectable Macro F1 scores (around 57–61%).

  Relevance: This demonstrates that fine-tuning is possible with a few hundred examples using techniques like LoRA, but performance may be limited. It sets a lower bound on data needs, suggesting our 30 examples might be insufficient even for LoRA, especially for complex enterprise queries.

- **LoRA on Larger SPARQL Corpora:** Schneider. (2024), evaluating work based on Sharma. (2025) [33], fine-tuned a LLaMA-7B model using LoRA on a much larger dataset of 30,000 conversational SPARQL QA examples (from SPICE and related sources). With this substantial training set, the model achieved high exact-match accuracy (75%–97%).

  Relevance: This study highlights the performance gains achievable with large-scale fine-tuning data (tens of thousands of examples), further emphasizing the gap between our available data (30) and the amount typically needed for state-of-the-art fine-tuned performance.

Given that our dataset size (30 examples) is significantly smaller than even the moderately sized dataset used by Mecharnia & d'Aquin and drastically smaller than the dataset used by Schneider., attempting fine-tuning was considered unlikely to yield robust results for our complex enterprise setting. This justified our decision to focus on prompting, RAG, and particularly agent-based approaches like ReAct, which rely less on large supervised datasets and more on in-context learning, retrieval, and iterative reasoning.

3.4 The Importance of Enterprise Datasets

Much research in Text-to-SQL and Text-to-SPARQL utilizes public benchmark datasets. While valuable, these may not fully capture the complexities of real-world enterprise environments. As noted by Sequeda. [32], the performance of LLMs on enterprise-specific questions remains relatively unexplored due to a lack of suitable benchmarks. They also highlight the under-investigated potential of KGs to provide essential business context for LLM-based question answering in these settings.

Relation to this thesis: Sequeda's work [32] directly motivates the context of this thesis. They identify a critical gap: the lack of evaluation of modern QA systems on realistic, complex enterprise data models. While their work provides a valuable starting point by using a subset (13 out of 199 tables) of an insurance data model, it still represents a simplification of a full enterprise schema. Our thesis aims to contribute to filling this gap by applying and evaluating Text-to-SPARQL techniques directly on a proprietary, complex enterprise KG. The use of this specific dataset, despite limiting the feasibility of fine-tuning, provides a more realistic testbed for evaluating the practical utility of methods like RAG and agentic frameworks in a setting where business context and schema complexity are significant challenges. This focus on an enterprise use case differentiates our work from studies relying solely on public benchmarks.

Furthermore, we drew inspiration from aspects of Google's proposed A2A (Agent2Agent Protocol) [35], particularly regarding flexible agent interactions. While we implemented some related concepts in our multi-agent solution, the fixed workflow inherent in our specific Text-to-SPARQL task meant that the broader dynamic capabilities envisioned by the A2A protocol were never fully utilized or necessary for this particular study. Mentioning it provides context on advanced agent paradigms considered,

even if not fully leveraged here.

3.5 Research GAP and summary

The review of literature spanning the history of natural language interfaces, the evolution of Text-to-SQL techniques, insights from recent surveys on LLM-based query generation [15], the challenges of fine-tuning in enterprise contexts [23, 33], and the critical need for evaluations on realistic enterprise datasets [32] reveals distinct opportunities for further research. Despite advancements, the practical application and comparative understanding of contemporary Large Language Model (LLM) techniques for Text-to-SPARQL generation within genuine enterprise environments highlight the following key gaps:

- **Limited Comparative Benchmarking of LLM Strategies for Enterprise Text-to-SPARQL:** There is a scarcity of comprehensive benchmarks that directly compare a diverse range of advanced LLM interaction patterns such as knowledge graph tailored Retrieval-Augmented Generation (RAG), structured Reasoning and Action (ReAct) frameworks, and multi-agent systems specifically for SPARQL generation within the complex constraints of real-world enterprise knowledge graphs, as opposed to evaluations predominantly on public datasets.
- **Insufficient Focus on Performance within Realistic Enterprise KG Settings:** Much of the current Text-to-SPARQL research utilizes public academic datasets, thereby not fully addressing the unique complexities (e.g., intricate schemas, specific query patterns) of proprietary enterprise KGs. Furthermore, with fine-tuning often being impractical in typical data-scarce enterprise scenarios (as detailed in Section 3.3), there remains a need for more research on LLM techniques that leverage in-context learning, effective schema utilization, and advanced agentic interactions.

This thesis is designed to address these identified gaps by undertaking a systematic evaluation of several leading LLM-based approaches for Text-to-SPARQL generation. The study focuses on their performance within a proprietary enterprise knowledge graph, using a comprehensive set of metrics.

Chapter 4

Method

This chapter outlines the scientific methodology employed to address the central research question of this thesis. It begins by restating the research question, then details the overall research strategy, including the role of the literature review and the experimental approach adopted. Subsequently, it describes the specific steps taken to compare different advanced Text-to-SPARQL methods, including the experimental setup, the distinct approaches evaluated, and the metrics used for comparison.

4.1 Research question

> **RQ:** *How do different advanced Text-to-SPARQL methods (Direct Prompting, RAG, ReAct and Multi-Agent) compare regarding accuracy, cost, response latency, and robustness within enterprise knowledge graph scenarios?*

4.2 Research Approach and Strategy

To answer the research question, this study adheres to a quantitative, experimental research methodology, grounded in the principles of the scientific method. This approach allows for the systematic comparison of different Text-to-SPARQL techniques by manipulating the method used (independent variable) and measuring its impact on key performance indicators (dependent variables: accuracy, cost, latency, and robustness) within a controlled enterprise knowledge graph environment. The experimental design ensures that observations are empirical and that comparisons are made systematically against defined benchmarks.

A focused literature review was foundational to this study. It served to establish the current state-of-the-art in Text-to-SPARQL generation, identify prevalent and emerging techniques, and understand existing evaluation methodologies. The insights gained from the literature review directly informed the selection of the advanced methods for comparison namely Direct Prompting, Retrieval-Augmented Generation (RAG), the ReAct framework, and a Multi-Agent system and highlighted the significance of the chosen performance metrics for enterprise scenarios.

The overall strategy to answer the RQ involves several interconnected stages, as depicted in Figure 4.1. Initially, a baseline performance is established using Direct

LLM Prompting. Subsequently, more advanced techniques (RAG, ReAct, and Multi-Agent systems) are implemented and evaluated. Each method is applied to a curated set of real-world enterprise queries against a representative knowledge graph. Performance data relating to accuracy, cost, latency, and robustness are collected for each approach. Finally, these metrics are systematically analyzed to compare the effectiveness and efficiency of the different methods. The subsequent sections in this chapter detail each phase of this research process.

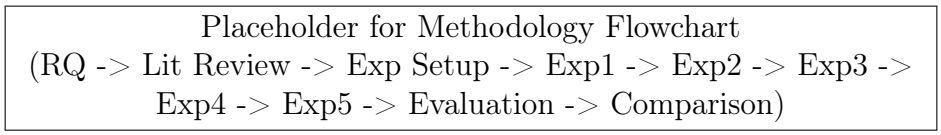

Figure 4.1: Flowchart of the Research Methodology.

4.3 Methodology

4.4 Literature Review Process Overview

To inform the experimental design and ensure grounding in existing knowledge, a focused literature review was conducted. The aim was to establish theory, map existing Text-to-SPARQL solutions, expose research gaps, and guide the selection of methods and evaluation strategies for *Benchmarking Approaches for QnA LLMs with SPARQL on EnterpriseData*.

4.4.1 Search Scope and Source Diversification

The literature search prioritized Google Scholar and IEEE Xplore due to their extensive coverage in relevant computer science and engineering fields. This was supplemented by exploring Text-to-SQL advancements (for transferable principles) and incorporating recommendations from [e.g., project advisor at in-parallel].

4.5 Environment Setup

The experiments were conducted on the primary computing device detailed below. All code execution, testing, and evaluations were performed locally on this machine.

4.6 Model settings and prompt details

The question answering systems we evaluated used a zero-shot prompt for all experiment on the LLM model GPT4.1-mini-2025-04-14 , that is instructed to generate a query, which is run against the database. Examples of prompt and question 1 in

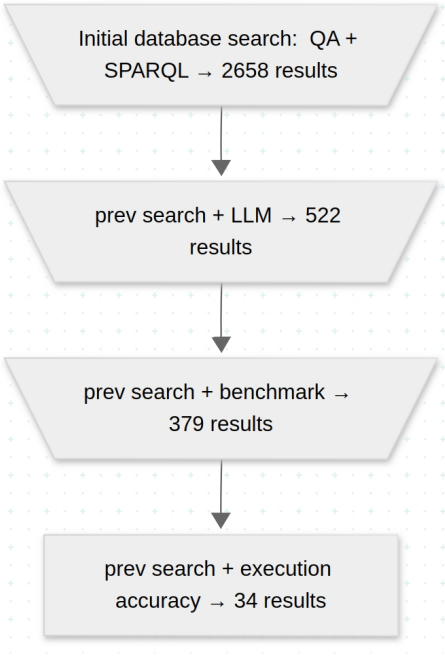

Figure 4.2: Flow diagram illustrating the literature screening and selection process, focusing on keywords such as "QA", "SPARQL", "LLM", "benchmark", and "execution accuracy".

Table 4.1: System specifications for the experimental environment

Specifications	**Details**
Model	Microsoft Surface Laptop 4
Processor	11th Gen Intel(R) Core(TM) i7-1185G7 @ 3.00GHz
GPU	Intel(R) Iris(R) Xe Graphics
Number of cores	4 (physical cores, 8 threads)
RAM	16 GB (16,216,936 kB total)
Storage	*Details not available in the provided report snippet*
Operating System	Ubuntu 22.04.5 LTS

Table 4.2: System specifications for the experimental environment (derived from report)

Appendix A and A.3 The particular parameters to the OpenAI API are just temperature set to 0.3 and Top-p was default of 1. We used GPT4.1-mini-2025-04-14 Because of the high cost of GPT4.5 or GPT4.1.

4.7 Key Python Libraries and Frameworks

This section details the primary Python libraries and frameworks utilized throughout the project, categorized by their main functionality. Version numbers indicate the minimum versions used or compatible with the experiments.

Core Web and API Frameworks

(Primarily for Experiment 4 - Multi-agent system)

- `fastapi` (version $\geq$0.109.0)
- `uvicorn` (version $\geq$0.27.0)
- `httpx` (version $\geq$0.26.0)

Environment and Configuration

(Used across all experiments)

- `python-dotenv` (version $\geq$1.0.0)
- `pydantic` (version $\geq$2.5.0)

Data Manipulation and Analysis

(Used extensively)

- `pandas` (version $\geq$1.5.0)
- `numpy` (version $\geq$1.21.0)

Progress Tracking and Utilities

- `tqdm` (version $\geq$4.64.0)

LangChain Ecosystem

(Used in Experiments 2, 3, 4, and 5)

- `langchain` (version $\geq$0.0.428)
- `langchain-core` (version $\geq$0.1.27)
- `langchain-community` (version $\geq$0.1.0)
- `langchain-openai` (version $\geq$0.1.0)

LangGraph for Multi-agent Systems

(Used in Experiment 4)

- `langgraph` (version ≥0.0.19)

LangSmith for Observability and used for token counting

(Used in Experiments 3 and 4)

- `langsmith` (version ≥0.0.3)

Vector Databases and Embeddings

(Used in Experiments 2 and 3)

- `faiss-cpu` (version ≥1.7.4)

RDF and SPARQL Support

(Core functionality across all experiments)

- `rdflib` (version ≥6.0.0)
- `SPARQLWrapper` (version ≥2.0.0)

Plotting and Visualization

(Used for generating report plots and tables)

- `matplotlib` (version ≥3.5.0)
- `seaborn` (version ≥0.11.0)

Jupyter Notebook Support

(Used for development and experimentation)

- `jupyter` (version ≥1.0.0)
- `ipykernel` (version ≥6.0.0)
- `notebook` (version ≥6.4.0)

Statistics and Data Science

(Used for generating tables and analysis)

- `scipy` (version ≥1.9.0)

HTTP Requests

(Used in multi-agent communication)

- `requests` (version ≥2.28.0)

Additional Utilities

(For JSON, regex, and text processing)

- `typing-extensions` (version ≥4.0.0)
- `regex` (version ≥2022.0.0)

Table 4.3: Rationale for Selection of Evaluated Text-to-SPARQL Methodologies

Methodology	**Key Capability / Rationale for Selection**	**Architecture Type**	**Perceived Real-World Applicability / Relevance**
Direct LLM Prompting	Establishes a performance baseline; represents the simplest approach to Text-to-SPARQL conversion by directly leveraging the LLM's inherent capabilities with full schema context.	Monolithic LLM call (LLM + Full Schema)	Common initial strategy for LLM tasks due to ease of implementation; useful for understanding raw LLM performance on a new task/dataset.
Retrieval-Augmented Generation (RAG)	Enhances LLM with dynamic, relevant context. Selection inspired by its efficacy highlighted in recent surveys (e.g., Hong et al., 2024[a]) for grounded QnA.	Retriever + LLM Generator (LLM + Retrieved Context)	Highly relevant for QnA over specific knowledge bases; addresses context limitations of LLMs and improves factual consistency.
Reasoning and Action (ReAct) Framework	Empowers LLM with agency for multi-step reasoning and tool use. A common and increasingly prominent agentic framework for complex problem-solving.	Iterative Agentic (LLM + Tools + Reasoning Loop)	Suitable for complex queries requiring intermediate steps, validation, or interaction with external systems; represents a sophisticated approach to agent design.
Multi-Agent Architecture	Decomposes tasks among specialized agents. Approach informed by trends in complex LLM systems (e.g., Hong et al., 2024[a]) aiming for modularity and robustness. But also the [6] covers multiagents for the use case of SPARQL generation	Orchestrated Collaborating Agents	Offers modularity and potential for specialized reasoning; applicable to complex workflows where different expertise is needed for sub-tasks.
Langchain Ontotext GraphDB QA Integration	Evaluates a pre-built solution using Langchain, a widely adopted framework for LLM application development. Langchain is a common choice due to its extensive integrations and abstractions for building complex chains and agents.[b]	Library-based Integration (Pre-defined Chain)	Represents the use of common, off-the-shelf tools; tests ease of integration and performance of established solutions in specific enterprise KG contexts.

[a] Assuming "Hong paper" refers to citations like Hong et al. (2024) mentioned in your thesis text (e.g., ‘ [14]‘ or ‘ [15]‘). Adjust citation style as needed.

[b] Langchain's popularity is evidenced by its large community, numerous integrations, and its frequent citation as a go-to framework for LLM application development (e.g., see LangChain's website, industry reports, and developer forums as of 2024-2025).

4.7.1 Experiment 1: Direct LLM Prompting (Baseline)

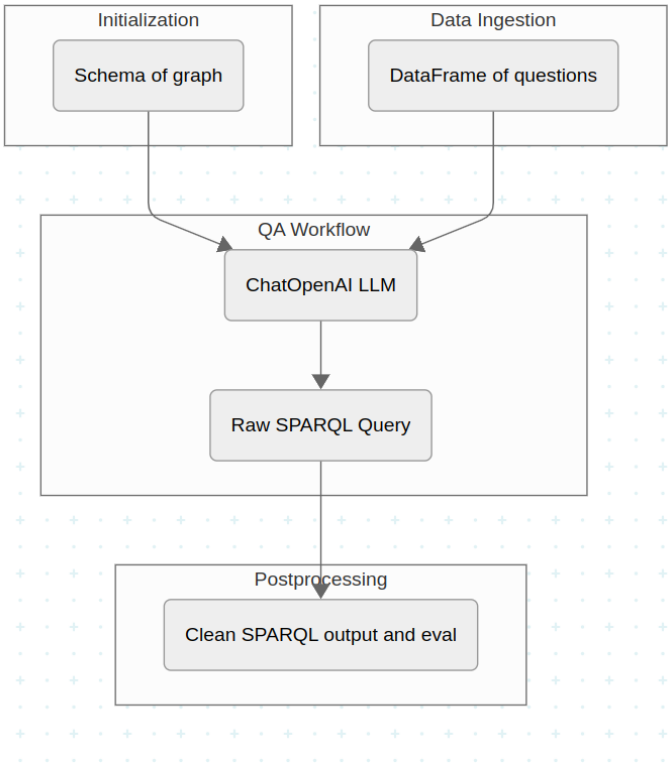

Figure 4.3: Conceptual diagram for Experiment 1: Direct LLM Prompting.

This first experiment establishes a baseline for Text-to-SPARQL conversion using direct prompting of a large language model (LLM). The objective is to measure the accuracy, latency, and computational cost (token usage) of a straightforward approach, providing a reference point for comparison with more advanced methods. We use this as a baseline because this is most people use LLMs as a service via (e.g, Openais chat platform)

1. **Environment and Model Setup:** The environment was configured with schema of the knowledge graph and the OpenAI API (The LLM).

2. **Query Generation Process:** Each natural language question was passed directly to the LLM with KG schema. The LLM was prompted with instructions to generate a SPARQL query capable of answering the question using the connected enterprise knowledge graph schema. The generated SPARQL query, the natural language answer produced by the chain, response latency, and token usage statistics were captured using openais `get_openai_callback`. prompr in appendix A.3

3. **Post-processing:** Generated SPARQL queries underwent a cleaning step to remove extraneous prefixes, formatting artifacts, or natural language explanations sometimes included by the LLM. The results were then saved to be later benchmarked. More information about the metrics and postprocessing these metrics can be found in section 4.8.

This baseline experiment provides essential reference data for evaluating the relative impact of the enhanced techniques implemented in subsequent experiments.

4.7.2 Experiment 2: RAG-Based Approach

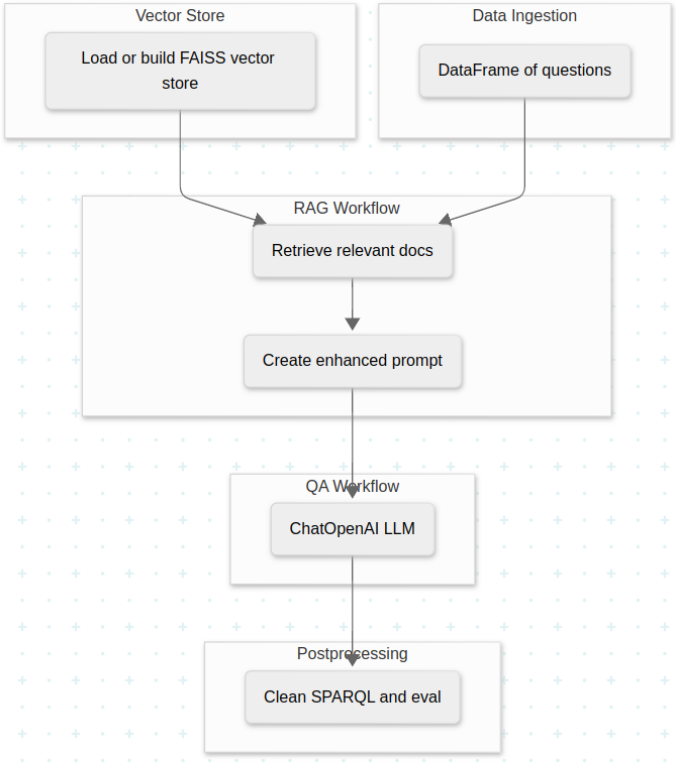

Figure 4.4: Conceptual diagram for Experiment 2: RAG-Based Approach.

The second experiment implemented a Retrieval-Augmented Generation (RAG) approach to potentially improve the accuracy and robustness of Text-to-SPARQL conversion. This method supplements the LLM's internal knowledge with context retrieved dynamically from an external knowledge source of the KG schema before generating the SPARQL query.

1. **Document Indexing and Vector Store Creation:** The KG schema were loaded and processed. They were split into manageable text chunks using a text splitter `RecursiveCharacterTextSplitter`. These chunks were then embedded using a suitable embedding model `OpenAI embeddings` and indexed into a vector store `langchain_community.vectorstores.FAISS`. Persistence was implemented, allowing the pre-computed vector store to be loaded if available, avoiding redundant processing on subsequent runs. The parameters used for the `RecursiveCharacterTextSplitter`

   (a) chunk_size = 600

   (b) chunk_overlap = 200

   (c) length_function = len

2. **Qurey generation:** we used a TOPK value of 15 for retrival and the prompt A.3 in the appendix.

3. **Post-processing:** Generated SPARQL queries underwent a cleaning step to remove extraneous prefixes, formatting artifacts, or natural language explanations sometimes included by the LLM. The results was then later saved to later benchmark. More information about the metrics and postprocessing these metrics can be found in section 4.8.

This RAG-based experiment aimed to assess whether providing targeted, dynamically retrieved context significantly improves SPARQL query generation performance compared to the direct prompting baseline.

4.7.3 Experiment 3: ReAct-Based Agent

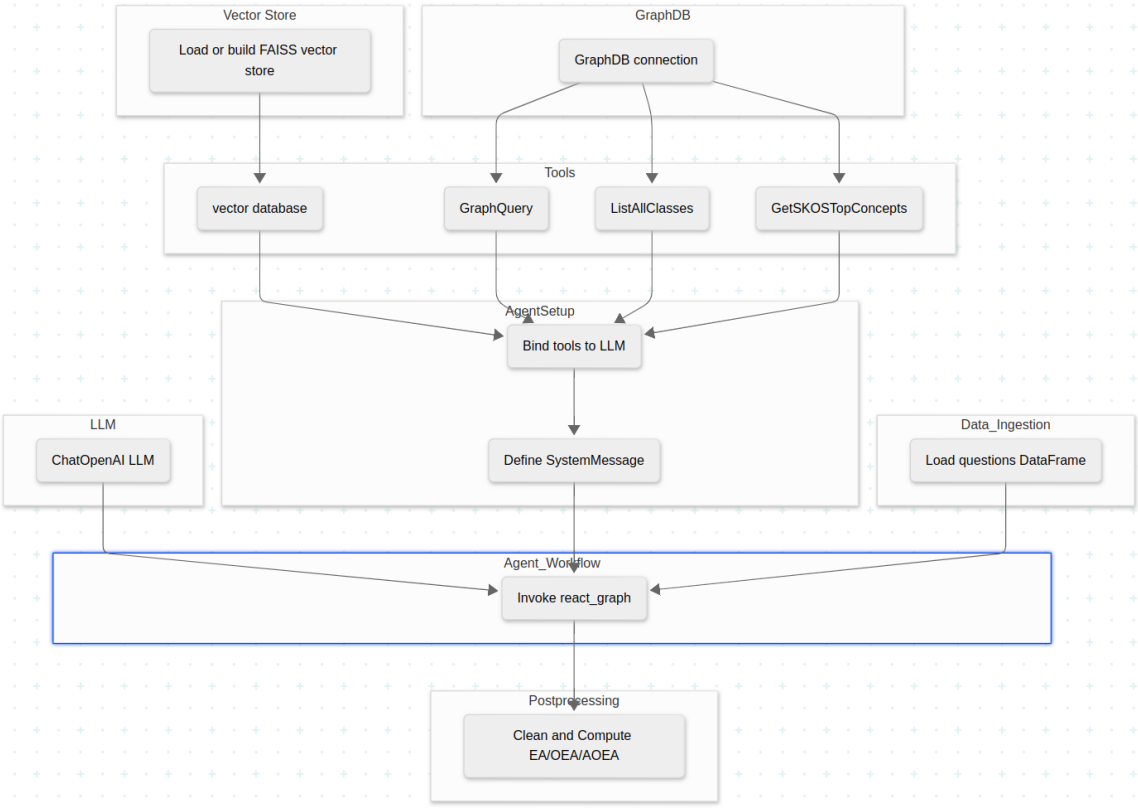

Figure 4.5: Conceptual diagram for Experiment 3: ReAct-Based Agent.

The third experiment implemented an agent architecture based on the ReAct (Reason + Act) [39] paradigm to enhance the Text-to-SPARQL system. This approach empowers the LLM to reason about a task, decide on actions (tool use), execute those actions, observe the results, and iteratively refine its approach until the final SPARQL query is constructed.

1. **Environment and Model Setup:** The environment utilized credentials for GraphDB and the OpenAI API. The LangChain and LangGraph libraries were

employed to construct the agent, integrating the LLM GPT4.1-mini-2025-04-14 with the GraphDB instance.

2. **Tool Definition and Integration:** A suite of tools was defined and made available to the agent. These tools included functions to:

   - Execute a given SPARQL query against Graph Database.
   - Do a vector search same as the one in experiment 2
   - Get all the classes in the core schema
   - Retrieve SKOS concepts or other specific graph structures.

   These tools were bound to the LLM, enabling it to invoke them programmatically based on its reasoning process.

3. **Agent Design (ReAct Pattern):** A system prompt guided the agent's behavior, instructing it to generate SPARQL queries to answer user questions and to utilize the provided tools iteratively if needed to explore the graph schema or refine its understanding. The agent framework facilitated the ReAct loop: the LLM reasons about the next step (thought), decides on an action (tool call or final answer), executes the action, and observes the result to inform the next reasoning step. the ReACT package we used is `AgentExecutor`, `create_react_agent` from `langchain.agents`. The prompt used can be found in appendix A.3

4. **Query Generation and Output Recording:** For each user question, the agent engaged in its reasoning-action loop, potentially invoking tools multiple times before producing a final SPARQL query. The system recorded the final generated answer, the final SPARQL query, overall latency for the process, and potentially intermediate steps or tool calls for analysis. Outputs were stored systematically.

5. **Post-processing:** The final generated SPARQL query underwent the standard cleaning procedure as in experiment 1 and experiment 2.

This ReAct-based agent experiment tested whether enabling the LLM to perform iterative reasoning and interact dynamically with the knowledge graph via tools leads to improved accuracy, robustness, or ability to handle more complex queries compared to the previous approaches.

4.7.4 Experiment 4: Multi-Agent System

The fourth experiment explored a modular multi-agent architecture for the Text-to-SPARQL task. This approach decomposes the problem into distinct sub-tasks, assigning each to a specialized agent. A central orchestrator coordinates the workflow between these agents, aiming to enhance robustness, maintainability, and potentially performance through specialization.

This experiment to inspiration from `https://github.com/GoogleCloudPlatform/`

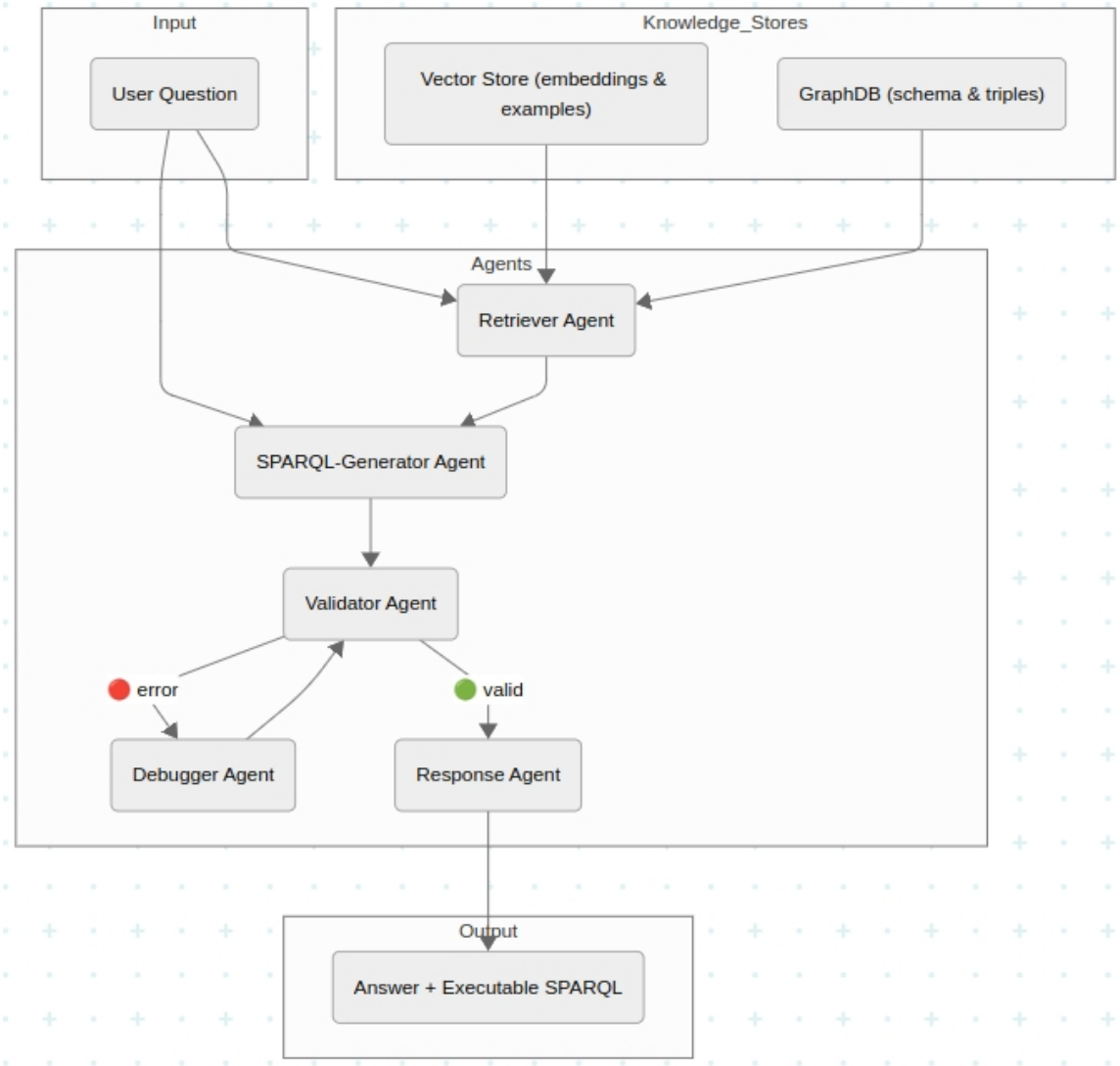

Figure 4.6: Conceptual diagram for Experiment 4: Multi-Agent System.

`Open_Data_QnA/`). As we did not find a good multi-agent solution for SPARQL we had to craft our own to test a multi-agent solution.

1. **System Architecture and Agent Roles:** The system was designed with multiple specialized agents, potentially implemented as independent services or functions, with defined roles: the prompts can be found in appendix: A.3

   - **Retriever Agent:** Responsible for fetching relevant context (from documents or KG) based on the input question. The retriever agent also follow the `create_openai_functions_agent` from `langchain.agents`. The Retriever Agent also has access to a vector store (Same as the one in the RAG) as an additional tool to use to retrieve information. Tool summary:
     - vector_search_tool
     - sparql_query_execution_tool
   - **Generator Agent:** Takes the question and retrieved context Generated by the Retriever Agent to produce an initial candidate SPARQL query.

- **Validator Agent:** Checks the generated query for syntactic correctness and potentially basic semantic validity against the schema. Returns a validation status (e.g., "OK" or specific error feedback).
- **Debugger Agent (Optional but recommended):** Attempts to automatically correct or refine the SPARQL query based on feedback from the Validator agent to prevent errors. Here a strict loop of maximum 3 iterations was implemented to reduce the model from getting stuck.
- **Executor Agent:** Executes a validated SPARQL query against the target GraphDB endpoint and returns the results. If executable the context is pass out from the model and if not okay passed back to the Debugger Agent with the error feedback.

2. **Orchestration Logic:** A controlling script or framework (the orchestrator) managed the interaction workflow between the agents, following a pipeline:
   2.1. Send question to Retriever Agent.
   2.2. Pass question and retrieved context to Generator Agent.
   2.3. Send generated query to Validator Agent.
   2.4. If validation fails, optionally send query and error feedback to Debugger Agent; send debugged query back to Validator (potentially in a loop with a maximum retry count).
   2.5. If validation succeeds, send the validated query to the Executor Agent.
   2.6. Return the final result from the Executor Agent.
3. **Error Handling and Robustness:** The orchestrator included logic for handling potential failures, such as agent unavailability (e.g., health checks before starting) or persistent validation/debugging failures. The validation step aimed to ensure only well-formed queries reach the execution stage.
4. **Output Recording:** Intermediate outputs from each agent (retrieved context, initial query, validation status, debugged query, final answer) along with timing information were logged for comprehensive analysis of the system's behavior and performance bottlenecks.

This multi-agent experiment investigated whether task decomposition and specialized agents coordinated via an orchestrator could yield improvements in system reliability, maintainability, transparency, or overall performance compared to monolithic or single-agent designs.

4.7.5 Exp 5 Ontotext GraphDB QA LangChain intergration

LangChain is an open-source framework designed to help developers build applications powered by large language models (LLMs) such as OpenAI's GPT-4. It provides a suite of tools and abstractions-available in Python and JavaScript-that make it easier to connect LLMs to external data sources, integrate them with other software components, and create complex, interactive AI applications like chatbots,

virtual agents, and retrieval-augmented generation (RAG) systems. This experiment is important as langchain has become one of the most common ways of connection LLMs with external sources such as graph databases.

The LangChain framework [12] and Ontotext GraphDB, specifically employing the OntotextGraphDBGraph connector and the `OntotextGraphDBQAChain` [19]. This integration facilitates Natural Language Querying (NLQ) of the knowledge graph, allowing questions phrased in natural language to be translated into SPARQL queries for execution against GraphDB.

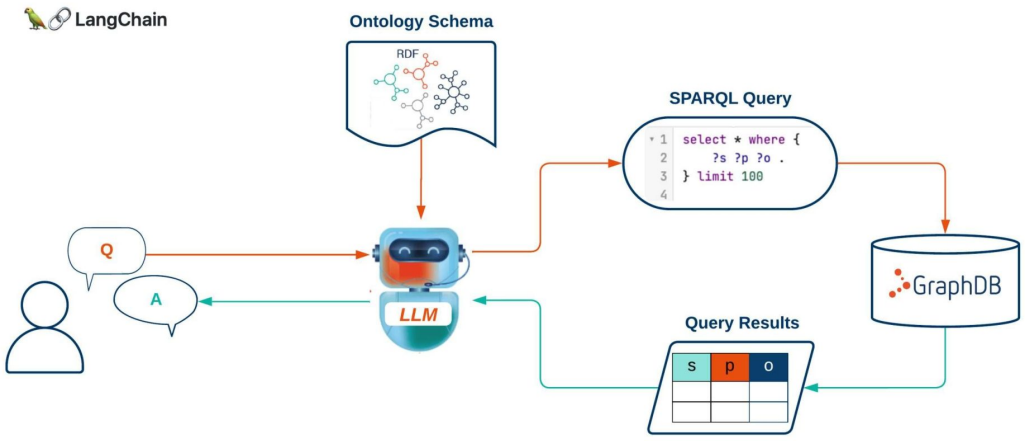

Figure 4.7: High-level workflow of the LangChain `OntotextGraphDBQAChain` used for natural language querying of Ontotext GraphDB. Image Adapted from [12]

- **Environment and Model setup:** The integration is fairly simple where the model accepts our ontology, graph connection and question as inputs.
- **Query Generation Process:** we also utilize the callback functionality of the `OntotextGraphDBQAChain` to capture our SPARQL Query used to answer the question. As the default output of this model is an Natural language answer based of a LLM observation of the executed query output. This model did not have a special prompt just the question feed into the model.
- **process results:** During the run we collect response latency for each function call, and token usage statistics were captured using openais `get_openai_callback`. These results was later saved for the benchmarking part.

The process, illustrated conceptually in Figure 4.7, typically involves the following steps when presented with a user's natural language question:

4.8 Evaluation

This section details the quantitative metrics and procedures used to evaluate and compare the performance of the four Text-to-SPARQL methods investigated (Direct

Prompting, RAG, ReAct, Multi-Agent). The evaluation focuses on objectively measuring key aspects relevant to practical application in enterprise scenarios: accuracy, response latency, computational cost, and robustness.

A crucial aspect of evaluating systems involving Large Language Models (LLMs) is addressing their inherent non-determinism; the same input query might yield different outputs across multiple executions. To account for this variability and obtain more reliable performance measures, each experiment was run 7 times for every natural language question in the benchmark dataset. This repetition allows for the calculation of average or overall metrics, providing a more robust assessment than single-run evaluations, particularly for accuracy, latency, cost, and robustness, as detailed below.

4.8.1 Accuracy

A common challenge in both Text-to-SQL and Text-to-SPARQL research is robust evaluation. Many studies rely on Execution Accuracy (EA), a standard metric prevalent in benchmarks like Spider [40] and BIRD [43] for Text-to-SQL, with analogous principles applied in Knowledge Graph Question and Answering (KGQA) contexts. Following the practice established in benchmarks such as Spider [40], we adopt Execution Accuracy (EA) as a fundamental measure.

Recognizing the non-deterministic nature of Large Language Models (LLMs), where the same input question might yield different generated queries across multiple runs, relying on a single EA score per question is insufficient. To account for this variability, we incorporate the concept of Overall Execution Accuracy (OEA). For each question in the benchmark, multiple generation attempts (runs) are performed. The OEA for that question is then calculated as the number of runs that achieved Execution Accuracy divided by the total number of runs.

```
OEA = (# of Accurate Executions (EA)) / (Total # of Runs).
```

An OEA of 1.0 (or 100%) signifies that every run produced a correctly executing query, while an OEA of 0.0 indicates that no run was successful for that specific question. If we get 1 Accurate Execution in 2 runs on a question we get an OEA of 50%

Finally, to provide a summarized performance measure across the entire benchmark or specific subsets of questions, we calculate the Average Overall Execution Accuracy (AOEA). This is simply the arithmetic mean of the OEA scores for all questions within the considered set.

$$\text{AOEA} = \frac{\sum_{i=1}^{N} \text{OEA}_{\text{question } i}}{\text{Total Number of Questions } (N)}$$

Therefore, for the evaluation conducted in this thesis, we utilize this hierarchy of

metrics: EA forms the basis for judging a single query's correctness; OEA quantifies the reliability of generating a correct query for a specific question given LLM non-determinism; and AOEA offers a high-level comparison of the overall performance of the different Text-to-SPARQL approaches evaluated. This methodology aligns with established practices while robustly handling the stochastic behavior inherent in the models under investigation.

4.8.2 Latency

We evaluate Latency, the time taken to generate a SPARQL query after receiving the input question. This reflects system responsiveness. We measure latency for each attempt (L_{run}). To account for variability, we compute the Average Latency per Question (AL_q) by averaging L_{run} across all R runs for that question. The Overall Average Latency (OAL) is then calculated by averaging AL_q across all N questions, providing an overall speed comparison:

$$AL_q = \frac{\sum_{r=1}^{R} L_{run,q,r}}{R} \tag{4.1}$$

$$OAL = \frac{\sum_{q=1}^{N} AL_q}{N} \tag{4.2}$$

4.8.3 Cost

Third, we evaluate Cost. To get a stable measure of computational resource usage (unlike changing API prices).However the token usage can be translated to the API cost in USD. We count tokens for each generation attempt. Specifically, we record three values for each run:

- **Prompt Tokens**: Size of the input to the model.
- **Completion Tokens**: Size of the generated SPARQL query.
- **Total Tokens**: The sum of prompt and completion tokens.

To calculate the average costs, we use the exact same calculation structure as we did for Latency. For each of the three token types above (Prompt, Completion, Total), we apply the two-step averaging process:

Applying this method yields our final cost metrics, such as Overall Average Prompt Tokens (OAPT), Overall Average Completion Tokens (OACT), and Overall Average Total Tokens (OATT). These metrics allow for a direct comparison of the token efficiency between the evaluated approaches.

4.8.4 Robustness

Robustness focuses on how many of the queries are actually able to be executable. Some quires can't be executed due to errors. This metric measures the approaches

different abilities to recognize potential errors and try to fix it. Robustness is calculated in the same way as Accuracy in section 4.8.1. Where an error means the query is not accurate and is calculated as 0 while an executable query is calculated as 1. Overall robustness is measured from all runs on this question and Average Overall robustness is measured on all the questions.

Chapter 5

Results and Analysis

This chapter presents the empirical findings from the benchmark experiments conducted to evaluate different Text-to-SPARQL generation methods, as outlined in Chapter 4. The primary goal is to address the central research question (RQ) concerning the comparative performance of Direct Prompting, Retrieval-Augmented Generation (RAG), the ReAct framework, and a Multi-Agent system within an enterprise knowledge graph scenario. The results focus on the Execution Accuracy (EA) metric, presented here as Overall Exact Answer (OEA) scores per question and averaged across questions (AOEA) for each experimental approach. Latency, scalability, and robustness aspects mentioned in the RQ may be discussed further in Chapter 6 based on any relevant data collected.

5.1 Results

The evaluation was performed using a curated set of 30 natural language questions paired with their gold-standard SPARQL queries, representative of queries encountered in the target enterprise environment. To maintain confidentiality, the specific questions were anonymised using numerical labels (1 through 30) due to the data not being public.

Four distinct experiments were conducted, each representing one of the core methodologies investigated:

- **Experiment 1 (exp1):** Direct LLM Prompting (Baseline), as described in Section 4.7.1.
- **Experiment 2 (exp2):** RAG-Based Approach, retrieving context before generation, as described in Section 4.7.2.
- **Experiment 3 (exp3):** ReAct-Based Agent, using iterative reasoning and tool use, as described in Section 4.7.3.
- **Experiment 4 (exp4):** Multi-Agent System, employing specialized collaborating agents, as described in Section 4.7.4.
- **Experiment 5 (exp5):** Langchain intergration, employing prebuilt graph database interaction system for the LLM as described in Section 4.7.5.

The subsequent sections present the results for Execution Accuracy (OEA and AOEA), Latency, Token Usage, and Robustness. Consistent formatting has been applied to all tables and figures as recommended.

5.1.1 Execution Accuracy

The Overall Exact Answer (OEA) scores achieved by each approach for each anonymised question, along with the overall Average OEA (AOEA), are summarised in Table 5.1. These scores represent the proportion of questions for which the generated SPARQL query produced results identical to the gold-standard query, following the adapted Spider-benchmark EA evaluation criteria discussed in Section 4.8. To provide a

Table 5.1: OEA Scores per Anonymised Question and AOEA per Experimental Approach

Question	Experimental Approach				
Label	exp1 (Direct)	exp2 (RAG)	exp3 (ReAct)	exp4 (Multi-Agent)	exp5 (Chain)
1	1.000	1.000	1.000	0.143	1.000
2	1.000	1.000	0.429	1.000	1.000
3	0.571	0.857	1.000	1.000	0.000
4	0.000	1.000	0.286	0.000	0.000
5	0.000	0.000	0.000	0.000	0.000
6	1.000	1.000	0.857	0.286	0.000
7	0.000	0.000	0.000	0.000	0.000
8	0.000	0.000	0.000	0.000	0.000
9	0.000	0.857	0.571	1.000	1.000
10	0.286	0.857	1.000	1.000	0.429
11	1.000	1.000	1.000	1.000	1.000
12	0.714	0.429	0.286	0.143	0.000
13	0.000	0.000	0.000	0.000	0.000
14	0.714	0.000	0.000	0.000	0.000
15	1.000	1.000	0.286	1.000	1.000
16	1.000	1.000	0.000	1.000	0.286
17	1.000	1.000	0.714	0.571	0.000
18	0.000	0.143	0.429	0.286	0.000
19	0.000	0.000	0.143	0.000	0.000
20	0.000	0.000	0.000	0.000	0.000
21	0.000	0.000	0.000	0.000	0.000
22	0.000	0.000	0.000	0.000	0.000
23	0.000	0.000	0.000	0.000	0.000
24	0.000	0.000	0.000	0.000	0.000
25	0.000	0.571	0.000	0.000	0.000
26	0.000	0.000	0.000	0.000	0.000
27	0.000	0.000	0.000	0.000	0.000
28	0.000	0.000	0.000	0.000	0.000
29	1.000	1.000	0.714	0.857	1.000
30	0.000	0.000	0.000	0.000	0.000
AOEA	**0.343**	**0.424**	**0.290**	**0.310**	**0.224**

meaningful visualization of the AOEA scores, a bar chart comparing the AOEA of the four experimental approaches would be effective (Figure 5.1).

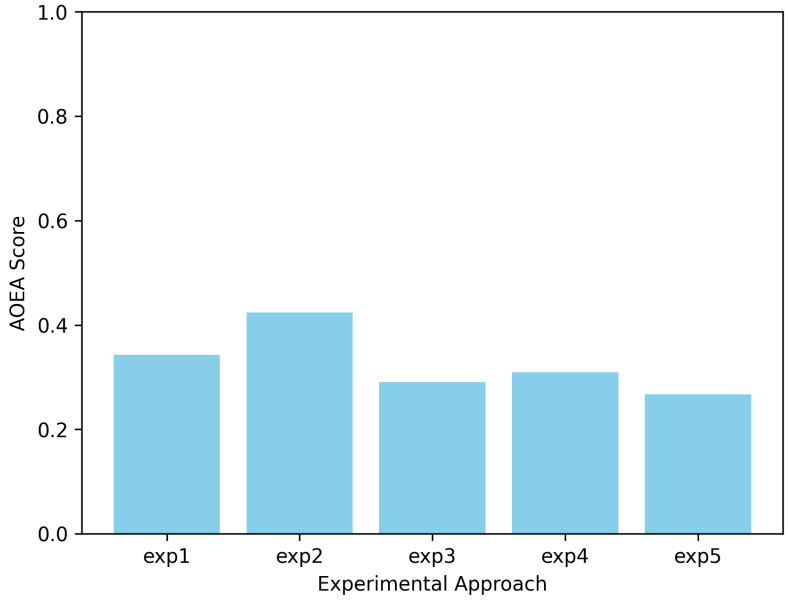

Figure 5.1: Comparison of Average Overall Exact Answer (AOEA) Scores by Experimental Approach.

5.1.2 Latency

The time taken (in seconds) by each approach to generate a SPARQL query for each anonymised question, along with the Overall Average Latency (OAL), is presented in Table 5.2. A box plot (Figure 5.2) illustrating the distribution of latencies for each experimental approach would effectively supplement this table, showing median, quartiles, and potential outliers.

5.1.3 Token Usage

The token usage for prompt and completion, along with the total tokens per question and the average total tokens for each experimental approach, is detailed in Table 5.3. A stacked bar chart (Figure 5.3) showing the average prompt and completion tokens for each experimental approach would offer a clear visual comparison of token efficiency.

5.1.4 Robustness

Robustness, in this context, is defined by the success rate of query execution. Table 5.4 shows the success (1 = executed, 0 = failed) for each question per approach. The caption notes "Overall robustness (robustness divided by run times per question)"; however, the table footer presents a "Success Rate," which is the average of these binary outcomes. For clarity, this section will focus on the "Success Rate" as the primary indicator of robustness from this table.

Use this. No filler:

—

Table 5.2: Latency in seconds per Anonymised Question and Overall Average Latency per Experimental Approach

Question	Experimental Approach				
Label	exp1 (Direct)	exp2 (RAG)	exp3 (ReAct)	exp4 (Multi-Agent)	exp5 (Chain)
1	3.396	3.546	3.522	10.383	5.670
2	3.797	3.261	37.932	11.757	8.794
3	2.945	2.583	17.834	11.027	5.446
4	3.262	2.782	61.721	9.698	5.266
5	4.729	2.750	42.260	7.817	23.045
6	2.525	2.264	11.921	17.044	4.943
7	3.373	3.093	25.953	14.816	8.098
8	2.747	2.359	46.699	10.049	5.507
9	2.791	2.330	24.331	9.652	38.618
10	3.016	2.402	17.296	10.697	7.030
11	3.363	2.919	2.219	17.889	7.851
12	3.161	3.559	27.269	11.334	7.339
13	4.466	4.256	58.376	11.957	4.765
14	3.108	3.008	15.404	10.154	4.763
15	4.090	3.913	14.628	17.560	88.653
16	4.161	3.153	1.924	11.098	10.288
17	3.072	2.403	12.883	10.809	5.295
18	2.629	2.211	3.418	8.595	2.696
19	4.084	2.924	12.709	9.205	6.172
20	3.591	3.364	20.059	11.532	4.403
21	3.515	3.184	46.451	11.817	5.670
22	3.496	2.878	82.703	11.918	4.006
23	2.588	2.406	2.943	12.562	3.925
24	4.003	4.044	32.980	11.134	6.710
25	3.915	2.989	23.961	10.020	10.753
26	3.811	5.938	52.312	9.154	4.154
27	2.920	2.779	2.445	11.018	3.861
28	3.305	2.659	2.353	15.089	5.195
29	3.467	2.784	7.365	7.489	6.980
30	3.556	4.276	29.123	10.119	3.892
OAL	**3.429**	**3.101**	**24.700**	**11.513**	**10.326**

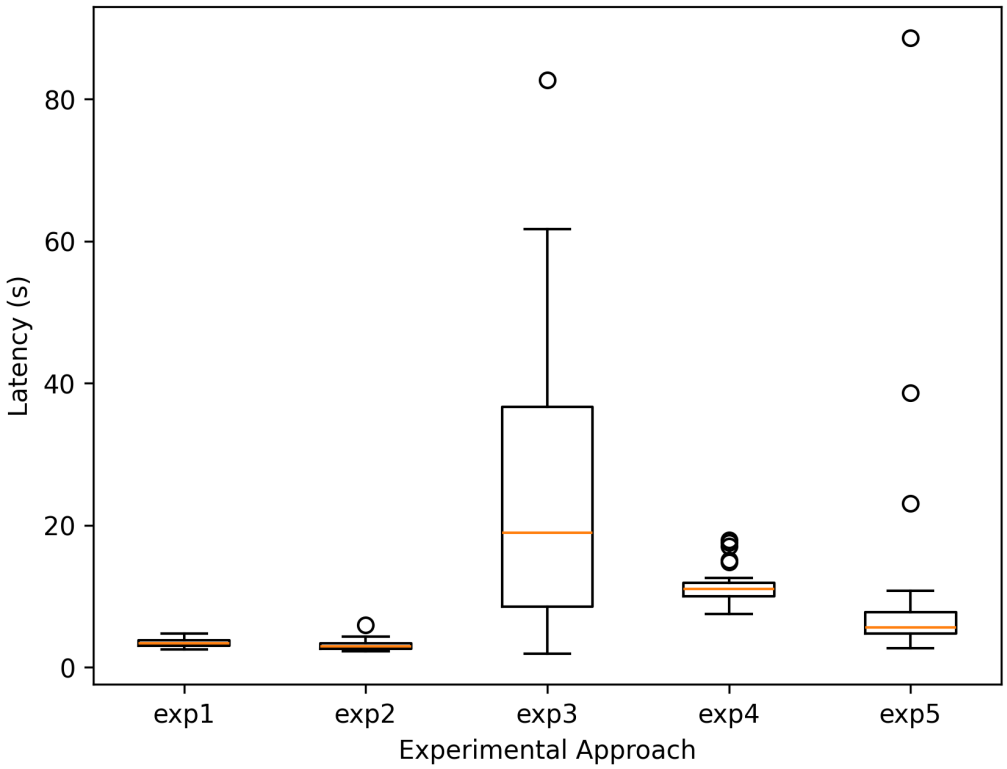

Figure 5.2: Distribution of Latency per Experimental Approach.

Table 5.3: Token Breakdown per Anonymised Question and Average Total Tokens per Experimental Approach

Q	exp1 (Direct)			exp2 (RAG)			exp3 (ReAct)			exp4 (Multi-Agent)			exp5 (chain)		
	prompt	completion	total	prompt	completion	total	prompt	completion	total	prompt	completion	total	prompt	completion	total
1	22588	109	22697	2060	97	2157	8733	1289	10022	978	478	1456	16278	228	16508
2	22594	82	22676	2531	79	2610	17187	767	17954	1409	387	1796	87215	212	87427
3	22584	69	22653	2993	40	3033	6022	472	6494	1390	394	1785	17055	226	17281
4	22589	98	22687	1895	102	1997	10662	1409	12071	1178	390	1568	16785	204	16989
5	22575	77	22652	2462	75	2537	20747	922	21669	1507	342	1849	19464	1224	20688
6	22573	72	22645	2088	61	2149	45145	525	45670	2189	564	2753	23102	149	23251
7	22594	106	22700	2274	106	2380	8856	1129	9984	1099	381	1480	16795	221	17016
8	22576	51	22627	2942	73	3015	498	70	568	791	269	1060	17925	192	18117
9	22585	48	22633	2921	65	2986	83516	1046	84561	1617	340	1956	20190	2174	22364
10	22592	113	22705	2322	94	2416	8350	953	9303	1249	432	1681	17647	315	17962
11	22592	97	22689	2181	94	2275	5283	972	6255	1928	523	2451	17481	396	17877
12	22598	78	22676	2948	83	3031	40428	5197	45625	1425	378	1803	21717	294	22011
13	22576	70	22646	2168	63	2231	499	79	578	1096	324	1420	17185	182	17367
14	22575	69	22644	2893	72	2965	497	67	564	1410	362	1772	16828	178	17006
15	22606	100	22706	2213	93	2306	7661	647	8307	1231	398	1629	27495	4891	32385
16	22613	82	22695	2492	87	2579	10862	1284	12147	1182	418	1601	16947	275	17222
17	22592	95	22687	2935	73	3008	1111	244	1354	1044	325	1369	17481	246	17727
18	22572	43	22615	1999	52	2051	4526	460	4986	1311	304	1615	16751	62	16813
19	22591	108	22699	1985	65	2050	4478	601	5079	1702	454	2156	17146	266	17413
20	22597	130	22727	1818	153	1971	20795	2370	23165	1569	586	2155	16801	172	16973
21	22586	101	22687	2166	96	2262	23893	2248	26141	926	418	1344	16779	211	16990
22	22578	35	22613	2643	48	2691	500	111	611	1408	353	1761	17273	165	17438
23	22594	70	22664	2258	56	2314	19614	645	20259	1562	386	1947	16795	153	16948
24	22574	59	22633	2780	59	2839	8378	576	8955	1448	263	1711	16993	235	17228
25	22606	150	22756	2222	172	2394	30369	2363	32732	955	481	1436	17316	496	17812
26	22568	59	22627	2767	66	2833	490	54	544	1418	280	1699	16775	149	16924
27	22586	33	22619	2905	33	2938	508	81	589	824	240	1064	18673	129	18802
28	22583	56	22639	2397	56	2453	506	98	604	1519	340	1859	31017	141	31158
29	22603	99	22702	2301	97	2398	7848	899	8747	1022	382	1404	17669	283	17952
30	22576	57	22633	2398	56	2454	13137	961	14098	802	275	1077	16759	125	16884
Avg	**22587**	**81**	**22668**	**2432**	**79**	**2511**	**13703**	**951**	**14655**	**1306**	**382**	**1689**	**20811**	**473**	**21284**

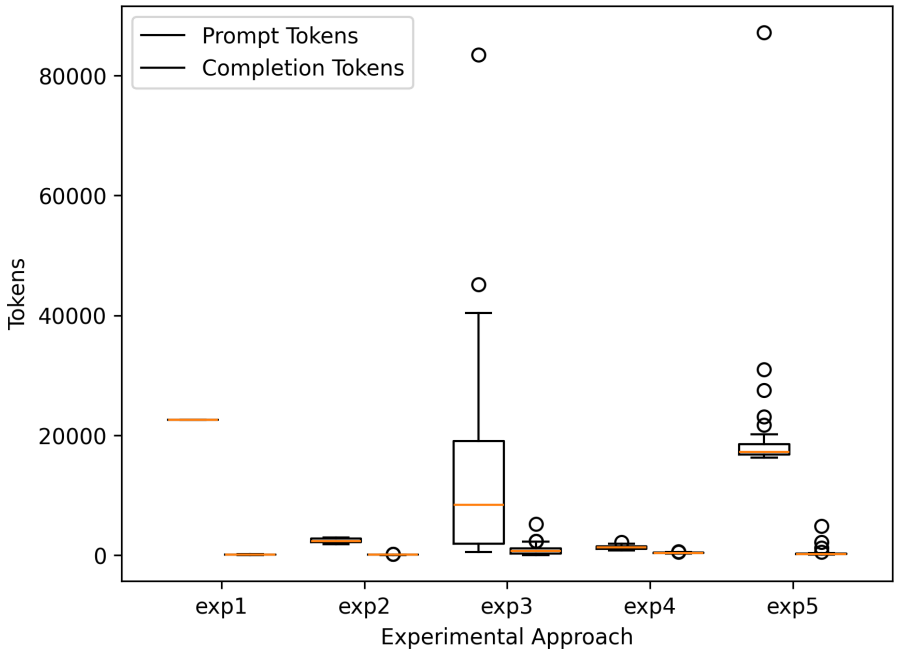

Figure 5.3: Average Prompt, Completion, and Total Tokens per Experimental Approach.

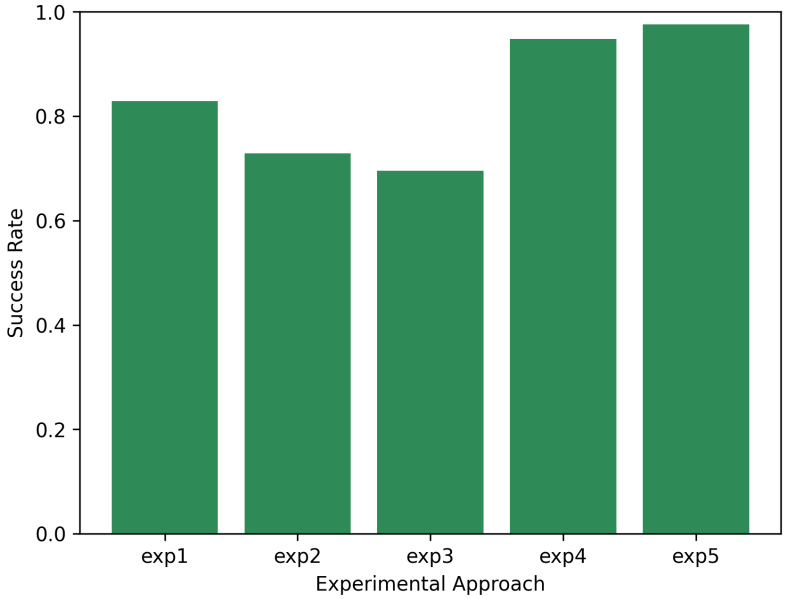

Figure 5.4: Comparison of Overall Success Rate (Robustness) by Experimental Approach.

Table 5.4: Robustness = Question: 1 = Executed, 0 = Failed, this table shows Overall robustness (robustness divided by run times per question)

Question	Experimental Approach				
Label	exp1 (Direct)	exp2 (RAG)	exp3 (ReAct)	exp4 (Multi-Agent)	exp5 (Chain)
1	1.00	1.00	1.00	1.00	0.86
2	1.00	1.00	0.86	0.86	1.00
3	1.00	0.71	0.29	1.00	1.00
4	1.00	0.00	0.71	1.00	1.00
5	0.86	0.57	1.00	1.00	1.00
6	0.57	0.86	1.00	1.00	1.00
7	1.00	1.00	1.00	1.00	1.00
8	1.00	1.00	1.00	1.00	1.00
9	1.00	1.00	0.71	1.00	1.00
10	1.00	1.00	1.00	1.00	1.00
11	1.00	1.00	0.00	0.71	1.00
12	1.00	0.86	0.86	1.00	1.00
13	1.00	0.57	1.00	1.00	1.00
14	0.86	0.00	0.14	1.00	1.00
15	1.00	1.00	0.57	0.86	1.00
16	1.00	1.00	0.00	0.00	1.00
17	0.00	0.43	0.29	1.00	1.00
18	0.00	1.00	0.57	1.00	1.00
19	0.00	0.00	0.86	1.00	1.00
20	1.00	1.00	1.00	1.00	1.00
21	0.29	0.86	1.00	1.00	1.00
22	0.29	0.43	0.57	1.00	1.00
23	1.00	1.00	0.43	1.00	1.00
24	1.00	0.86	1.00	1.00	1.00
25	1.00	0.57	1.00	1.00	1.00
26	1.00	0.00	0.71	1.00	1.00
27	1.00	1.00	0.29	1.00	1.00
28	1.00	1.00	0.71	1.00	0.43
29	1.00	0.14	0.71	1.00	1.00
30	1.00	1.00	0.57	1.00	1.00
Success Rate	**0.83**	**0.73**	**0.70**	**0.95**	**0.98**

5.1.5 Comprehensive Statistical Summary (Std 95% CI)

Table 5.5 summarizes the key metrics for all methods with their standard deviation, coefficient of variation, and 95% confidence intervals based on 210 runs per method.

Standard deviation (Std) and coefficient of variation (CV) show the spread and relative variability. The 95% confidence intervals (CI) show the uncertainty around each mean and make it clear if differences between methods are likely to be due to chance.

Table 5.5: Descriptive statistics and 95 % confidence intervals for each metric (210 runs per experiment).

Metric	Experiment	Mean	Std	CV	CI_{lower}	CI_{upper}	N
	Direct LLM	0.343	0.476	1.39	0.278	0.408	210
	RAG-Based	0.424	0.495	1.17	0.356	0.491	210
EA	ReAct Agent	0.290	0.455	1.57	0.229	0.352	210
	Multi-Agent	0.310	0.463	1.50	0.246	0.373	210
	QA-Chain	0.224	0.418	1.87	0.167	0.281	210
	Direct LLM	3.43	1.24	0.36	3.26	3.60	210
	RAG-Based	3.10	1.27	0.41	2.93	3.27	210
Latency (s)	ReAct Agent	24.70	43.83	1.77	18.74	30.66	210
	Multi-Agent	11.45	4.82	0.42	10.79	12.10	210
	QA-Chain	7.77	7.22	0.93	6.78	8.75	210
	Direct LLM	22 667.73	36.82	0.0016	22 662.72	22 672.74	210
	RAG-Based	2 510.73	342.05	0.136	2 464.20	2 557.26	210
Total tokens	ReAct Agent	14 654.55	37 716.56	2.57	9 523.66	19 785.44	210
	Multi-Agent	1 688.64	686.35	0.406	1 595.27	1 782.01	210
	QA-Chain	26 637.01	13 178.36	0.495	24 844.25	28 429.77	210

5.2 Summary Metrics

The following table summarizes the performance metrics for the various approaches applied to the Questions

Table 5.6: Performance comparison for Question 1.

Method	Latency (s) (OAL)	Accuracy (AOEA)	Total Tokens	Robustness
Direct LLM	3.429	0.343	22668	0.83
Vector RAG	3.101	0.424	2511	0.73
ReAct	24.700	0.290	13703	0.70
Multi-Agent	11.513	0.310	1306	0.95
Langchain Chain	10.326	0.224	20811	0.98

5.3 Analysis

This section provides an objective analysis of the results presented in Tables 5.1, 5.2, 5.3, and 5.4, focusing on the comparative performance of the five experimental approaches based on the collected data.
Scope. Tables 5.1–5.4 benchmark five Text-to-SPARQL strategies on accuracy, speed, token cost, and execution success.

5.3.1 Execution Accuracy

- **Rankings (AOEA).** RAG (exp2) **0.424** > Direct (exp1) 0.343 > Multi-Agent (exp4) 0.310 > ReAct (exp3) 0.290 > Langchain (exp5) 0.224.
- **Patterns.**
  - Easy Q1: all perfect except exp4.
  - Hard Q7: all zero.
  - RAG shines on Q2, 4, 6, 9–11, 15–17, 29.
  - Direct is solid on Q1, 2, 6, 11, 15–17, 29 but erratic elsewhere.
  - Multi-Agent excels on Q3, 9–11, 15, 16, 29 thanks to specialist agents.
  - ReAct peaks on Q1, 3, 6, 10, 11 yet drops on the rest.
  - Chain scores only on a handful (Q1, 2, 9, 11, 15, 29).
  - Q5, 7, 8, 13, 20–24, 26–28, 30 stump every method.

5.3.2 Latency

- **Rankings (OAL).** RAG 3.10 s < Direct 3.43 s < Chain 7.77 s < Multi-Agent 11.51 s < ReAct 24.70 s.
- **Observations.**
  - RAG's retrieval adds negligible delay vs. Direct.
  - Chain shows extreme outliers (Q9 38.6 s, Q15 88.7 s).
  - ReAct latency is volatile (e.g. Q4, 13, 22) due to multi-step reasoning.
  - Multi-Agent overhead stems from agent coordination.

5.3.3 Token Usage

- **Average total tokens.** Multi-Agent 1 689 < RAG 2 511 < ReAct 14 655 < Chain 26 637 < Direct 22 668.
- **Drivers.**
  - Direct always feeds the full schema (˜22 600 prompt tokens) high but stable cost.

  - RAG trims context; variance comes from overlapping chunks.
  - ReAct loops can explode usage (up to 15 tool cycles).
  - Multi-Agent limits TOP-K to 5 and often skips tools lowest cost.
  - Chain sometimes loads huge contexts (Q2 87 427 tokens).

5.3.4 Robustness (Success Rate)

- **Rankings.** Chain 0.98 > Multi-Agent 0.95 > Direct 0.83 > RAG 0.73 > ReAct 0.70.
- **Notes.**
  - Chain fails only on Q1 (0.86) and Q28 (0.43).
  - Multi-Agent misses solely Q16.
  - RAG's misses (Q4, 14, 19, 26) show execution fragility despite high accuracy.
  - ReAct's iterative steps lead to many failures (Q3, 11, 16).
  - Remember: a runnable query can still be wrong (high robustness $\neq$ high OEA).

It is important to note the distinction between Execution Accuracy (OEA) and Robustness (Success Rate). A query can be successfully executed (robustness = 1 or close to 1) but still yield incorrect results (OEA < 1.0). Conversely, if a query fails to execute (robustness = 0), its OEA is inherently 0 for that attempt.

5.4 Summary of Results

This chapter presented the empirical results from evaluating five Text-to-SPARQL generation methods across Execution Accuracy (EA), Latency, Token Usage, and Robustness.

Key comparative findings include:

- **Execution Accuracy (AOEA):** The RAG-based approach (exp2) demonstrated the highest average execution accuracy (0.424 AOEA), followed by Direct Prompting (exp1) at 0.343. Langchain integration (exp5) had the lowest AOEA (0.224).
- **Latency (OAL):** The RAG-based approach (exp2) was the fastest on average (3.101s OAL), closely followed by Direct LLM Prompting (exp1 at 3.429s). Langchain integration (exp5) was moderate (10.326s), while ReAct (exp3) was significantly slower.
- **Token Usage (Avg. Total Tokens):** The Multi-Agent system (exp4) was the most token-efficient (average 1689 tokens). Langchain integration (exp5) (21284 tokens) and Direct LLM Prompting (exp1) (22668 tokens) were the least efficient due to large prompt sizes.

- **Robustness (Success Rate):** Langchain integration (exp5) exhibited the highest robustness with a 98% success rate in query execution, closely followed by the Multi-Agent system (exp4) at 95%.

No single method consistently outperformed others across all metrics. For instance, the RAG approach, while best for accuracy and latency, had a lower robustness score than the Langchain integration and Multi-Agent systems. The Langchain integration, best for robustness, had the lowest accuracy and was among the higher token consumers. The Multi-Agent system, second best for robustness and best for token efficiency, had a moderate accuracy score. These trade offs highlight the varying strengths and weaknesses of each approach in the context of an enterprise knowledge graph scenario. The implications of these findings and a more detailed interpretation will be explored in Chapter 6.

5.5 Summary Metrics

The following table summarizes the performance metrics for the various approaches applied to the Questions.

Table 5.7: Performance comparison for Question 1.

Method	**Latency (s) (OAL)**	**Accuracy (AOEA)**	**Total Tokens**	**Robustness**
Direct LLM	3.429	0.343	22668	0.83
Vector RAG	3.101	0.424	2511	0.73
ReAct	24.700	0.290	13703	0.70
Multi-Agent	11.513	0.310	1306	0.95
Langchain Chain	10.326	0.224	20811	0.98
Summary (Average)	**10.614**	**0.327**	**12200**	**0.84**

5.6 Rank Summary Metrics

The following table summarizes the winner scores for the various approaches.

Table 5.8: Winner comparison for Question 1 with Experiment Averages.

Method	**Latency**	**Accuracy**	**Total Tokens**	**Robustness**	**Average Rank**
Direct LLM	2	2	5	3	**3.00**
Vector RAG	1	1	2	4	**2.00**
ReAct	5	4	3	5	**4.25**
Multi-Agent	4	3	1	2	**2.50**
Langchain Chain	3	5	4	1	**3.25**

Chapter 6

Discussion

This chapter synthesizes these findings, interprets their significance in relation to the research question, discusses the limitations encountered during the study, and reflects on the broader implications of the work. The aim is not merely to reiterate the analysis from Chapter 5, but to engage in a deeper argumentation about what these results signify, to consider different perspectives, and to place the findings within the broader academic and practical context.

6.1 Addressing the Research Question

The central research question (RQ) guiding this investigation was:

> **RQ:** *How do different advanced Text-to-SPARQL methods (Direct Prompting, RAG, ReAct and Multi-Agent Architectures) compare regarding accuracy, cost, response latency, and robustness within enterprise knowledge graph scenarios?*

The experimental results presented in Chapter 5 provide the empirical basis for answering this question by comparing Direct LLM Prompting (exp1), Retrieval-Augmented Generation (RAG) (exp2), a ReAct-Based Agent (exp3), a Multi-Agent System (exp4), and a Langchain integration (exp5) across these four key dimensions. The following synthesis declares the principal answers derived from the findings.

6.1.1 Comparative Performance Analysis Summary

6.1.1.1 Accuracy

The RAG-based approach (exp2) demonstrated the highest accuracy (0.424 AOEA). This finding suggests that dynamically providing targeted, relevant KG schema/instance snippets is more effective than presenting the LLM with the entire schema (as in Direct Prompting, 0.343 AOEA) or relying on more complex reasoning chains with the current implementations (ReAct at 0.290, Multi Agent at 0.310). The Langchain integration (exp5) showed the lowest accuracy (0.224). The formulation of less ambiguous benchmark questions, as discussed in Section 6.3, may have contributed to RAG's success by facilitating precise context retrieval. During the thesis some results on question that was less ambiguous the ReACt model seemed to perform the best but this is someting that could be researched further.

6.1.1.2 Cost (Token Usage)

The Multi Agent system (exp4) was the most cost effective (1689 tokens on average), likely due to its modularity and focused agent prompts. RAG (exp2) also offered good token economy (2511 tokens). Direct Prompting (exp1) and the Langchain integration (exp5) were significantly more expensive (22668 and 21284 tokens, respectively). ReAct (exp3) had high and variable costs (14655 tokens). There are a lot of factors that play a role here. As discussed in section **??**. Some values include like TOPK value of 15 or the parameters when vectorizing. The prompt in the ReAct model. But also parameter setting in the ReAct model could be optimized and the results might prove better.

6.1.1.3 Response Latency

RAG (exp2) and Direct Prompting (exp1) exhibited the lowest response latencies (3.101s and 3.429s OAL, respectively). Agentic systems (ReAct exp3: 24.700s; Multi-Agent exp4: 11.513s) and the Langchain integration (exp5: 10.326s) were slower, reflecting their more involved processing. Here a high token usage often correlates with a high latency. Here some more work on setting limits on how long a prompt can be to minimize the token usage on some question that are larger. a outline for 1 questions in ReAct took 8.6 minutes while other question only needed 1 second.

6.1.1.4 Robustness

The Langchain integration (exp5) and the Multi-Agent system (exp4) demonstrated the highest robustness (98% and 95% execution success rates, respectively), indicating strong capabilities in generating syntactically valid queries. RAG (exp2), despite its accuracy, showed lower robustness (73%). Robustness is not correlated with the right answer as we can see with the results.

6.1.2 Synthesis: Answering the Research Question

The evidence indicates that no single method universally excelled.

- **RAG (exp2)** emerged as a leading contender for balancing accuracy, latency, and cost, though its robustness warrants attention.
- **The Multi-Agent System (exp4)** showed strength in cost-efficiency and robustness but with moderate accuracy.
- **Direct Prompting (exp1)** provided a baseline but was prohibitively expensive.
- **ReAct (exp3)**, in its current configuration, underperformed, highlighting its sensitivity to toolset quality and potentially question type.
- **The Langchain integration (exp5)** offered high robustness but at the expense of accuracy and cost for this specific KG.

Therefore, the answer to the RQ is nuanced: the optimal Text to SPARQL method for an enterprise KG scenario depends on the specific prioritization of accuracy, cost, latency, and robustness.

This section moves beyond the summary of results to delve into the interpretation of these findings, engaging in argumentation and relating them to existing knowledge and expectations.

6.1.3 The Ascendancy of RAG: Focused Context vs. Comprehensive Knowledge

The superior accuracy and efficiency of the RAG approach (exp2) compared to Direct Prompting (exp1) is a central finding. While Direct Prompting provided the LLM with the entirety of the schema, RAG's strategy of selectively retrieving relevant schema parts (top-15 results) proved more effective. This supports the "less is more" principle in the context of LLM limitations with extensive contexts, as highlighted by Liu. [22] regarding the "lost in the middle" phenomenon. One might argue that an LLM with access to the full schema *should* theoretically outperform one with partial context. However, the practical reality appears to be that current LLMs struggle to effectively navigate and utilize vast, undifferentiated information dumps. RAG's retrieval step acts as an effective pre filter, focusing the LLM's attention.

An interesting point for discussion is RAG's moderate robustness (73%) despite its high accuracy. On one hand, this could imply that when RAG's retrieval provides highly relevant context, the LLM generates semantically correct queries. On the other hand, if the retrieval is slightly off or if the integration of retrieved snippets into the prompt is imperfect, it might lead to syntactically flawed queries more often than systems with built in validation loops (like exp4) or highly regularized outputs (potentially exp5). This trade off between semantic precision driven by good context and syntactic robustness deserves further investigation.

6.1.4 The Cost of Agency: ReAct and Multi Agent Systems

The Multi Agent system (exp4), while more successful in token efficiency and robustness, still showed only moderate accuracy. This could be interpreted in several ways. Perhaps the decomposition of tasks, while beneficial for efficiency (e.g., specialized, smaller prompts for each agent), led to information loss or suboptimal synthesis by the generator agent. Alternatively, the validator and debugger agents, while enhancing robustness, might not have been sophisticated enough to correct more nuanced semantic errors, only syntactic ones. The balance between the benefits of modularity and the potential for accumulated errors or miscommunications in a multi agent chain is a critical consideration. Here also the use of different prompts and giving the retriever agent more tools might make it increase the cost.

6.1.5 Pre-built Integrations: Robustness vs. Specificity

The Langchain integration (exp5) showcased excellent robustness but the lowest accuracy. This finding might reflect a common characteristic of off the shelf tools: they

are often designed for broad applicability and error prevention, potentially sacrificing nuanced accuracy for a specific, proprietary KG unless significant customization is undertaken. The high token cost, similar to direct prompting, suggests it also attempts to process a large portion of the schema, but its internal query construction mechanisms might be more generalized. This raises the question: for enterprise KGs, is it more effective to build a tailored solution (like RAG or a custom agent) or to adapt a generic but robust framework? The answer likely depends on available development resources and the tolerance for accuracy deviations.

The low performance could be that the integration is focused on Natural Language Querying (NLQ). So maybe it's focused to better handle question that are more human like and less ambiguous. As this is how the market the product. They also mention in their documents [12] Sometimes the LLM may generate a SPARQL query with missing prefixes, syntactic or other errors. In such cases, we try to amend this by prompting the LLM to correct the query a certain number of times. In addition to the invalid query, we also provide the LLM with the error message and the ontology schema.

6.1.6 Unexpected Findings and Conflicting Data

One potentially unexpected pattern was the divergence between accuracy and robustness for RAG (high accuracy, moderate robustness) versus the Langchain integration (low accuracy, high robustness). This highlights that these are distinct qualities; a system can reliably produce executable queries that are nonetheless incorrect, or it can often produce correct queries but sometimes fail syntactically. This distinction is crucial for enterprise deployment decisions.

6.2 Reflections on Methodological Choices and Observations

Several factors relating to the experimental setup and execution warrant discussion as they provide context to the results.

6.2.1 Prompting Strategies

A key aspect of this study was the evaluation of different *approaches* to Text-to-SPARQL generation, rather than a strict comparison of identical prompts. Each method (Direct Prompting, RAG, ReAct, Multi-Agent) inherently requires a different prompting architecture to leverage its specific mechanisms. For instance, RAG prompts are designed to incorporate retrieved context, while ReAct prompts must guide the thought-action-observation loop. While the core instructions to the LLM regarding the task (generating SPARQL) and desired output style were kept consistent where possible (e.g., persona, tone), the structural differences in prompts are an intrinsic part of what distinguishes the methodologies. Future work could explore fine-grained prompt optimization within each architectural approach, but the goal here was to assess the broader strategies.

6.2.2 Influence of LLM Model

The experiments were conducted using the GPT4.1-mini-2025-04-14 model. As noted from informal tests, more advanced models like "GPT 4.5" (hypothetical or a future version) yielded significantly better results across all approaches. This indicates that the performance ceilings observed are specific to the chosen LLM, and advancements in foundational models will likely elevate the performance of all evaluated techniques. The relative strengths and weaknesses between methods, however, might still hold, though this would require further comparative studies with newer models.

6.2.3 The TOP-K Parameter

The RAG system (exp2) and the retriever tool in the Multi-Agent system (exp4) utilized a TOP-K value of 15 for context retrieval. This parameter is crucial: it dictates the volume of retrieved data samples (e.g., schema snippets, examples) provided to the LLM. A lower TOP-K value would generally reduce token costs and latency but could negatively impact accuracy if critical information is missed. Conversely, a higher value might increase costs and latency, and potentially introduce noise. The chosen value of 15 represents a balance based on preliminary exploration, but was not exhaustively optimized for every query type. The Multi-Agent system's retriever used a fraction of this (effectively 5), which contributed to its token efficiency but might have limited the context for its generator agent.

6.2.4 Execution Accuracy (EA)

As EA has limitations. Its strictness can penalize valid alternative queries yielding the correct information but differing trivially in format see Example of EA Limitations below. Furthermore, potential inconsistencies in benchmark ground truths can affect score reliability. Consequently, alternative metrics have been proposed, such as Component Matching (CM) [40] for logical form correctness, Valid Efficiency Score (VES) [43] combining accuracy with query runtime, and LLM-based evaluation frameworks like FLEX [18] aiming for human-like judgment with a LLM as A judge.

FLEX [18] served as a interesting alternative but we found it less transparent, as LLM as a judge reasoning may be a bit opaque. As this is a academic study to evaluate LLMs we left this EVAL metric off the table. But the benchmark is less stricter.

The BIRD paper (VES metric paper) [43] found that reliance on Execution Accuracy (EA) has been critically analyzed in recent research, particularly regarding its limitations in real-world evaluation scenarios. A 2024 arXiv study provides the most comprehensive examination of these issues [18].

For the benchmarking conducted in this thesis (as detailed in Chapter 3), Execution Accuracy (EA) was selected as the primary metric due to its widespread use and objective nature. Also we wanted to use EA and measure latency (runtime for each query seprettly). VES Could serve as an alternative that Combines execution accuracy (EA) with runtime efficiency. As our aim with this thesis is to be as close to the

real world as possible and use enterprise data VES could serve as a more important alternative when working with data in scale but for now we believe the metric just make the results more confusing.

In our paper we use the Spider-benchmark style for EA eval we don't care about order (not row order or col order). Since the Original spider-benchmark is used for SQL a revised version the spiderbenchmark is adopted to handle the syntax of SPARQL instead of SQL in our paper. There is another benchmark called SPIDER4SPARQL but the public repository was in JAVA so we had to build our own spider style benchmark from scratch in python. This covers most of the good from the CM metric that does not care about the order either. But the CM metric can give partial right to a EVAL if for example only 50% of the rows are right. This is not something we want as that 50% could prove important. This is why Spider-benchmark EA was adopted for this thesis.

Example of EA Limitations

Question: List all employee names and their departments.
Answer:

Employee Name	Department
Bob	HR
Alice	Sales

The following answer is accurate under Execution Accuracy (EA):

Department	Employee Name
Sales	Alice
HR	Bob

even though the column headers, the orders of the column and the orders of the rows are different. The answer is accurate.

However, the following answer is inaccurate:

Emp_Name	Department
"Alice"	Sales
"Bob"	HR

Emp_Name (className)	Employee Name	Department
"bob"	Bob	HR
"Alice"	Alice	Sales

because the column header `Emp_Name` does not exactly match `Employee Name`, even though the data is correct. Sometimes Emp_Name can serve a different col but contain the same value and because we ask for a name and not what type of name it retrieves. Even though the model contains the right data the models still scores 0 points.

6.3 Limitations of the Study

Beyond the methodological reflections above, this study has several limitations:

- **Question Formulation and Ambiguity:** The 30 benchmark questions were formulated to reduce ambiguity, aiming for a clear comparison of core query generation capabilities. This may have favored methods like RAG that benefit from precise signals for vector search. As observed informally with a more colloquial and ambiguous dataset, the ReAct agent (exp3) showed potential in handling such ambiguity, a strength not fully tested by the current benchmark. Real world performance on inherently vague user queries might thus differ.

- **ReAct Agent Toolset Enhancement:** The toolset for the ReAct agent (exp3) was foundational. A more comprehensive suite of tools for schema exploration, entity linking, query validation, and iterative refinement could significantly elevate its performance. This was a primary constraint on exp3's potential observed in this study.

- **Dataset Size and Generalizability:** The evaluation used a proprietary enterprise KG and a limited set of 30 question to query pairs. While offering realism, findings might not directly generalize to other KGs with different schemas, data characteristics, or query complexities. The data scarcity also precluded fine-tuning as a comparative method.

- **Single LLM Evaluation:** Findings are specific to the GPT4.1-mini-2025-04-14 model. Performance characteristics can vary significantly with other LLMs.

6.4 Summary and Position of this Thesis

This chapter reviewed related work primarily in Text-to-SQL and the emerging field of LLM-based Text-to-SPARQL generation. We traced the historical development, highlighting how foundational Text-to-SQL research informs current approaches. We examined key LLM-based strategies identified in recent surveys [15], including direct prompting, RAG, agent-based systems, and fine-tuning. The discussion emphasized the data requirements for fine-tuning [23, 33] and the challenges associated with enterprise datasets [32].

Several research gaps emerge from this review:

- While Text-to-SQL is mature, the application of advanced LLM techniques (especially sophisticated agentic frameworks like ReAct [39]) to Text-to-SPARQL is less explored.

- Most evaluations use public benchmarks, leaving performance on complex, real-world enterprise KGs relatively unknown [32].

- There is limited investigation into how techniques perform in data-scarce enterprise scenarios where fine-tuning with thousands of examples is not feasible.

6.5 Contribution to Closing Research Gaps

This thesis aimed to address specific gaps identified in the review of related work (Chapter 3):

- **Limited Comparative Benchmarking for Enterprise Text-to-SPARQL:** This research provides a direct comparative benchmark of diverse LLM interaction patterns (Direct Prompting, RAG, ReAct, Multi-Agent, and a Langchain integration) specifically for SPARQL generation on a real-world enterprise KG. This extends evaluations often confined to public datasets or fewer methods.
- **Insufficient Focus on Realistic Enterprise KG Settings and Data Scarcity:** By using a proprietary, complex enterprise KG and a limited dataset (30 pairs), this work offers insights into method performance where extensive fine tuning is impractical. This addresses the need for research on LLM techniques leveraging in context learning and advanced agentic interactions in such data scarce scenarios.

The empirical results and the analysis of trade offs between accuracy, cost, latency, and robustness contribute practical knowledge for industry practitioners and a foundation for future research in applying LLMs to enterprise KGs. While Text to SQL is more mature, this work helps advance understanding in the specific domain of Text to SPARQL within an industrial context.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

This thesis embarked on an investigation to evaluate and compare the performance of distinct contemporary methodologies for generating SPARQL queries from natural language questions. The study was specifically situated within the context of a complex enterprise knowledge graph, aiming to provide empirically grounded insights into the practical suitability of Direct LLM Prompting, Retrieval Augmented Generation (RAG), ReAct based agents, a Multi Agent system, and a pre built Langchain integration.

The empirical findings presented in Chapter 5 reveal a nuanced performance landscape. The RAG-based approach (exp2) achieved the highest average execution accuracy (0.424 AOEA) and demonstrated the lowest average latency (3.101s OAL), suggesting that providing targeted, retrieved context significantly aids the LLM. However, its robustness (73% success rate) was moderate. Conversely, the Langchain integration (exp5) and the Multi-Agent system (exp4) exhibited superior robustness, with execution success rates of 98% and 95% respectively. The Multi Agent system also proved to be the most token efficient (average 1689 tokens). Direct LLM Prompting (exp1), while achieving the second-highest accuracy (0.343 AOEA), incurred the highest token costs (average 22668 tokens) due to the inclusion of the entire schema in the prompt. The ReAct based agent (exp3), in its current configuration, generally underperformed across accuracy, cost, and latency, indicating that its effectiveness is highly contingent on the sophistication of its toolset and the nature of the queries. No single method emerged as universally superior across all evaluated metrics (accuracy, latency, cost, and robustness).

To ensure robustness of the conclusions drawn from this study, a comprehensive statistical analysis was performed. Standard deviations, coefficients of variation, and 95% confidence intervals (Table 5.5) confirm that the observed performance differences between methods are statistically meaningful and not simply due to chance. This statistical evidence reinforces the empirical findings: notably, the superior accuracy and latency performance of the RAG-based method, and the efficiency advantages of the Multi-Agent system. These results provide a strong statistical backing for practitioners deciding on the optimal Text-to-SPARQL method based on enterprise-specific operational priorities and constraints.

The answer to the research question is therefore multifaceted. The RAG methodology presents a compelling balance of accuracy and speed for enterprise Text-to-SPARQL tasks, though its syntactic reliability needs consideration. For scenarios prioritizing robustness and cost-efficiency, a Multi-Agent architecture shows significant promise. The pre built Langchain integration offers high reliability in query execution but at the cost of accuracy and token usage for this specific enterprise KG. Direct Prompting, while simple to implement, is not cost effective for large schemas. The choice of an optimal approach is thus contingent upon the specific operational priorities and constraints of the enterprise environment, such as the relative importance of semantic accuracy versus guaranteed execution, or speed versus computational budget.

The main contribution of this thesis lies in the systematic, comparative benchmarking of these contemporary LLM based techniques applied directly to a proprietary, complex enterprise knowledge graph. This work provides valuable empirical evidence regarding their practical performance trade-offs, particularly in a data scarce context where extensive model fine-tuning is often impractical. By evaluating these methods beyond standard public benchmarks, this research contributes to closing the identified gap concerning the understanding of how such advanced systems perform under realistic enterprise conditions, characterized by intricate schemas and specific query patterns. The findings offer actionable insights for practitioners seeking to implement Text-to-SPARQL solutions and a documented foundation for future academic inquiry in this domain.

This study is, however, subject to certain limitations. The evaluation was performed using a specific set of 30 question query pairs tailored to a single enterprise knowledge graph and conducted with the GPT4.1-mini-2025-04-14 LLM; these results may not directly generalize to other KGs, query types, or foundational models. The formulation of benchmark questions, aimed at reducing ambiguity, might have favored certain retrieval-based methods. Furthermore, the toolset for the ReAct agent was foundational, and its potential may not have been fully realized. The primary evaluation metric was Execution Accuracy, which, while objective, has known limitations in capturing all aspects of query correctness or semantic equivalence.

7.2 Future Work

The findings and limitations of this thesis illuminate several promising avenues for future research and development in the field of Text-to-SPARQL generation for enterprise knowledge graphs:

- **Enhanced Agent Toolsets and Reasoning:** Future work should focus on significantly enhancing the toolsets for agent-based systems like the ReAct agent (Experiment 3). This includes developing more sophisticated tools for granular schema exploration, dynamic entity linking, query validation against semantic constraints (beyond syntax), and more advanced iterative query refinement and self-correction capabilities. For the ReAct agent, efforts could concentrate on refining the sophistication and granularity of its toolset, investigating tools for validating specific SPARQL clauses, executing partial queries

for intermediate feedback, or more seamlessly integrating RAG capabilities (as explored in Experiment 2) as a dynamic information retrieval tool within the ReAct reasoning loop. Exploring different agent architectures and reasoning paradigms within ReAct, as well as the impact of fine-tuning LLMs specifically for ReAct based Text-to-SPARQL generation (drawing inspiration from studies like FireACT by Chen [3]), presents another promising direction.

- **Multi-Agent System Development:** The Multi-Agent system (Experiment 4) offers considerable scope for further experimentation. Future iterations could involve equipping individual specialized agents with their own tailored toolsets, thereby enhancing their autonomy and problem solving capabilities. While this thesis drew inspiration from the Open Data QnA architecture, which is primarily designed for Text-to-SQL, constraints limited the current implementation to core concepts. A future research endeavor could undertake a more comprehensive adaptation and implementation of the Open Data QnA framework for Text-to-SPARQL.

- **Hybrid Approaches:** Investigating hybrid methodologies that combine the strengths of different approaches could be beneficial. For example, integrating the robust validation mechanisms of the Multi-Agent system (Experiment 4) or the Langchain integration (Experiment 5) with the accurate context retrieval of RAG (Experiment 2) might lead to a system that is both accurate and robust.

- **Advanced RAG Techniques:** Optimizing chunking strategies and embedding models for KG schemas is also a key area. Specifically for the RAG approach (Experiment 2), further investigation is warranted into diverse vectorization strategies. As Han, Liu, and Wang [10] detail, the optimal selection and application of embedding techniques (e.g., machine learning models, word embeddings, feature extraction algorithms) for KG schema and content merit deeper exploration. Additionally, a comparative analysis of vectorization targets, such as vectorizing the KG schema versus vectorizing a large set of questions to SPARQL query pairs, is needed to determine their relative efficacy.

- **Impact of LLM Advancements:** Re-evaluating the presented methodologies using newer and more capable Large Language Models (e.g., GPT 4.5 or subsequent versions beyond the GPT4.1-mini-2025-04-14 model is crucial. As foundational models improve, the performance ceilings and relative strengths of these Text-to-SPARQL techniques may shift. Comparative analysis of models differing in size, architecture, or pre training data could reveal which are best suited for the complex reasoning and syntax generation required in enterprise contexts.

- **Test different models from different companies** The experiment could be tested with different models on the same data to compare results with different models from different companies, such as Antropics Claudee or Google's gemini. As this was not covered in the scope of this thesis,s this is something that could be addressed in further work and research.

- **Prompt Engineering and Optimization:** Conducting a more systematic exploration of prompt engineering techniques tailored to each specific methodology and the Text to SPARQL task could unlock further performance gains. This includes investigating the efficacy of various prompting styles such as few-shot learning, chain of thought prompting, and self critique mechanisms customized to SPARQL syntax and KG structures to optimize the performance of all evaluated approaches.

- **Broader and More Diverse Benchmarks:** Expanding the evaluation to include larger and more diverse sets of questions, covering a wider range of query complexities and ambiguities, would enhance the generalizability of the findings. Developing benchmarks that include more colloquial and less precisely formulated user queries would better reflect real-world usage and test the ambiguity handling capabilities of methods like ReAct.

- **Alternative Evaluation Metrics:** While Execution Accuracy is a standard, future research should incorporate a broader suite of evaluation metrics. This could include component matching, semantic similarity scores, Valid Efficiency Score (VES) for combined accuracy and runtime, or even qualitative human evaluations to assess the intuitiveness and correctness of generated queries from a user perspective. Exploring LLM as a judge frameworks like FLEX, while ensuring transparency, could also be considered.

- **Parameter Optimization:** A systematic optimization of key parameters within each method, such as the TOP-K value in RAG and agentic retrieval tools, or the iteration limits and specific prompts within agent loops, could lead to more refined performance characteristics.

- **Scalability and Cost Benefit Analysis for Enterprises:** Future studies could delve deeper into the scalability of these approaches for very large enterprise KGs and conduct a more comprehensive cost benefit analysis that considers development effort, maintenance, and infrastructure costs alongside runtime performance.

- **Handling Ambiguity and User Interaction:** Research into how these systems handle ambiguous natural language questions and how they could engage in clarifying dialogues with users to resolve ambiguity before attempting SPARQL generation would be highly valuable for practical enterprise deployments.

By addressing these areas, future research can continue to advance the capabilities of Text-to-SPARQL systems, making complex enterprise data more accessible and actionable for a broader range of users.

References

[1] Anthropic, "Prompt engineering interactive tutorial," https://github.com/anthropics/prompt-eng-interactive-tutorial, 2023, accessed: 2025-04-19.

[2] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," 2020, arXiv:2005.14165 [cs.CL], version 4. [Online]. Available: https://doi.org/10.48550/arXiv.2005.14165

[3] B. Chen, C. Shu, E. Shareghi, N. Collier, K. Narasimhan, and S. Yao, "Fireact: Toward language agent fine-tuning," 2023.

[4] D. A. Dahl, M. Bates, M. Brown, W. Fisher, K. Hunicke-Smith, D. Pallett, C. Pao, A. Rudnicky, and E. Shriberg, "Expanding the scope of the ATIS task: The ATIS-3 corpus," in *Human Language Technology: Proceedings of a Workshop held at Plainsboro, New Jersey, March 8-11, 1994*, 1994. [Online]. Available: https://aclanthology.org/H94-1010/

[5] D. Edge, H. Trinh, N. Cheng, J. Bradley, A. Chao, A. Mody, S. Truitt, D. Metropolitansky, R. O. Ness, and J. Larson, "From local to global: A graphrag approach to query-focused summarization," 2021, version 4, revised 12 Apr 2021. [Online]. Available: https://arxiv.org/abs/2005.11401

[6] N. El Abid Amrani, M. Youssfi, O. E. K. Abra, and O. Bouattane, "A novel deep learning approach for semantic interoperability between heterogeneous multi-agent systems," *Engineering, Technology & Applied Science Research*, vol. 9, no. 4, pp. 4566–4573, 2019. [Online]. Available: http://www.etasr.com/index.php/ETASR/article/view/2872

[7] C. Finegan-Dollak, J. K. Kummerfeld, L. Zhang, K. Ramanathan, S. Sadasivam, R. Zhang, and D. Radev, "Improving text-to-SQL evaluation methodology," in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, I. Gurevych and Y. Miyao, Eds. Melbourne, Australia: Association for Computational Linguistics, Jul. 2018, pp. 351–360. [Online]. Available: https://aclanthology.org/P18-1033/

[8] Google Cloud Platform, "Open data qna: Conversational data access with llms on google cloud," https://github.com/GoogleCloudPlatform/Open_Data_QnA, 2025, accessed: 2025-04-25, includes architecture, usage guides, and deployment options.

[9] C. C. Green and B. Raphael, "The use of theorem-proving techniques in question-answering systems," in *Proceedings of the 1968 23rd ACM national conference*. ACM, 1968, pp. 169–181. [Online]. Available: https://doi.org/10.1145/800186.810578

[10] Y. Guo, M. Guo, J. Su, Z. Yang, M. Zhu, H. Li, M. Qiu, and S. S. Liu, "Bias in large language models: Origin, evaluation, and mitigation," *arXiv preprint arXiv:2411.10915*, 2024.

[11] Y. Han, C. Liu, and P. Wang, "A comprehensive survey on vector database: Storage and retrieval technique, challenge," 2023, submitted on 18 Oct 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2310.11703

[12] N. Hateva, "Natural language querying of graphdb in langchain," https://www.ontotext.com/blog/natural-language-querying-of-graphdb-in-langchain/, 2024, updated November 18, 2024.

[13] G. G. Hendrix, E. D. Sacerdoti, D. Sagalowicz, and J. Slocum, "Developing a natural language interface to complex data," *ACM Transactions on Database Systems (TODS)*, vol. 3, no. 2, pp. 105–147, 1978. [Online]. Available: https://doi.org/10.1145/320251.320253

[14] Z. Hong, Z. Yuan, Q. Zhang, H. Chen, J. Dong, F. Huang, and X. Huang, "Next-generation database interfaces: A survey of llm-based text-to-sql," 2024, arXiv:2406.08426 [cs.CL], version 1, 12 Jun 2024. [Online]. Available: https://arxiv.org/abs/2406.08426

[15] ——, "Next-generation database interfaces: A survey of llm-based text-to-sql," 2024, last revised 13 Mar 2025 (v5). [Online]. Available: https://doi.org/10.48550/arXiv.2406.08426

[16] E. J. Husom, A. Goknil, L. K. Shar, and S. Sen, "The price of prompting: Profiling energy use in large language models inference," *arXiv preprint arXiv:2407.16893*, 2024.

[17] S. Iyer, I. Konstas, A. Cheung, J. Krishnamurthy, and L. Zettlemoyer, "Learning a neural semantic parser from user feedback," in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, R. Barzilay and M.-Y. Kan, Eds. Vancouver, Canada: Association for Computational Linguistics, Jul. 2017, pp. 963–973. [Online]. Available: https://aclanthology.org/P17-1089/

[18] H. Kim, T. Jeon, S. Choi, S. Choi, and H. Cho, "Flex: Expert-level false-less execution metric for reliable text-to-sql benchmark," 2024, preprint, under review. Version 4, revised 28 Oct 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2409.19014

[19] LangChain, "Ontotextgraphdbqachain – langchain python api reference (langchain-community v0.3.22)," https://python.langchain.com/api_reference/community/chains/langchain_community.chains.graph_qa.ontotext_graphdb.OntotextGraphDBQAChain.html, 2024.

[20] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W. tau Yih, T. Rocktäschel, S. Riedel, and D. Kiela, "Retrieval-augmented generation for knowledge-intensive nlp tasks," in *Advances in Neural Information Processing Systems (NeurIPS)*. Curran Associates, Inc., 2020. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2020/hash/6b493230205f780e1bc26945df7481e5-Abstract.html

[21] F. Li and H. V. Jagadish, "Constructing an interactive natural language interface for relational databases," *Proc. VLDB Endow.*, vol. 8, no. 1, p. 73–84, Sep. 2014. [Online]. Available: https://doi.org/10.14778/2735461.2735468

[22] N. F. Liu, K. Lin, J. Hewitt, A. Paranjape, M. Bevilacqua, F. Petroni, and P. Liang, "Lost in the middle: How language models use long contexts," *Transactions of the Association for Computational Linguistics (TACL)*, vol. 11, pp. 1158–1176, 2023, arXiv:2307.03172 [cs.CL]. [Online]. Available: https://doi.org/10.48550/arXiv.2307.03172

[23] T. Mecharnia and M. d'Aquin, "Performance and limitations of fine-tuned LLMs in SPARQL query generation," in *Proceedings of the Workshop on Generative AI and Knowledge Graphs (GenAIK)*, G. A. Gesese, H. Sack, H. Paulheim, A. Merono-Penuela, and L. Chen, Eds. Abu Dhabi, UAE: International Committee on Computational Linguistics, Jan. 2025, pp. 69–77. [Online]. Available: https://aclanthology.org/2025.genaik-1.8/

[24] Ontotext. (2025) What is rdf? Accessed: 2025-04-19, published in Ontotext Knowledge Hub - Fundamentals. [Online]. Available: https://www.ontotext.com/knowledgehub/fundamentals/what-is-rdf/

[25] ——. (2025) What is sparql? Accessed: 2025-04-19, published in Ontotext Knowledge Hub - Fundamentals. [Online]. Available: https://www.ontotext.com/knowledgehub/fundamentals/what-is-sparql/

[26] OpenAI, J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, R. Avila, I. Babuschkin, S. Balaji, V. Balcom, P. Baltescu, H. Bao, M. Bavarian, J. Belgum, I. Bello, J. Berdine, G. Bernadett-Shapiro, C. Berner, L. Bogdonoff, O. Boiko, M. Boyd, A.-L. Brakman, G. Brockman, T. Brooks, M. Brundage, K. Button, T. Cai, R. Campbell, A. Cann, B. Carey, C. Carlson, R. Carmichael, B. Chan, C. Chang *et al.*, "Gpt-4 technical report," 2023, arXiv:2303.08774 [cs.CL], version 6. [Online]. Available: https://doi.org/10.48550/arXiv.2303.08774

[27] F. Petroni, T. Rocktäschel, S. Riedel, P. Lewis, A. Bakhtin, Y. Wu, and A. Miller, "Language models as knowledge bases?" in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Hong Kong, China: Association for Computational Linguistics, 2019, pp. 2463–2473. [Online]. Available: https://www.aclweb.org/anthology/D19-1250

[28] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:160025533

[29] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," 2019. [Online]. Available: https://arxiv.org/abs/1910.10683

[30] P. Rathle, "Graphrag: Leveraging graphs in llm systems," https://www.youtube.com/watch?v=8oVvcaNPVrE&t=3432s, Neo4j, 2024, youTube, accessed on 2025-04-19.

[31] A. Roberts, C. Raffel, and N. Shazeer, "How much knowledge can you pack into the parameters of a language model?" 2020. [Online]. Available: https://arxiv.org/abs/2002.08910

[32] J. Sequeda, D. Allemang, and B. Jacob, "A benchmark to understand the role of knowledge graphs on large language model's accuracy for question answering on enterprise sql databases," arXiv, Technical Report 2311.07509, 2023, submitted on 13 Nov 2023, 34 pages. [Online]. Available: https://doi.org/10.48550/arXiv.2311.07509

[33] A. Sharma, L. Lara, A. Zouaq, and C. J. Pal, "Reducing hallucinations in language model-based sparql query generation using post-generation memory retrieval," 2025, arXiv:2502.13369 [cs.CL], version 1, licensed under CC BY-SA 4.0. [Online]. Available: https://doi.org/10.48550/arXiv.2502.13369

[34] A. Singhal. (2012) Introducing the knowledge graph: things, not strings. Google Blog. Accessed: 2025-04-19. [Online]. Available: https://blog.google/products/search/introducing-knowledge-graph-things-not-strings/

[35] R. Surapaneni, M. Jha, M. Vakoc, and T. Segal. (2025) Announcing the agent2agent protocol (a2a): A new era of agent interoperability. Published on April 9, 2025, Google Developers Blog. Accessed: 2025-04-25. [Online]. Available: https://developers.googleblog.com/en/a2a-a-new-era-of-agent-interoperability/

[36] L. R. Tang and R. J. Mooney, "Automated construction of database interfaces: Intergrating statistical and relational learning for semantic parsing," in *2000 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*. Hong Kong, China: Association for Computational Linguistics, Oct. 2000, pp. 133–141. [Online]. Available: https://aclanthology.org/W00-1317/

[37] W. A. Woods, "Transition network grammars for natural language analysis," *Communications of the ACM*, vol. 13, no. 10, pp. 591–606, 1970. [Online]. Available: https://doi.org/10.1145/355598.362773

[38] N. Yaghmazadeh, Y. Wang, I. Dillig, and T. Dillig, "Type- and content-driven synthesis of sql queries from natural language," 2017, arXiv:1702.01168 [cs.DB]. [Online]. Available: https://doi.org/10.48550/arXiv.1702.01168

[39] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "React: Synergizing reasoning and acting in language models," *arXiv preprint arXiv:2210.03629*, 2022.

[40] T. Yu, R. Zhang, K. Yang, M. Yasunaga, D. Wang, Z. Li, J. Ma, I. Li, Q. Yao, S. Su, Z. Zhang, and D. Radev, "Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, 2018, pp. 3911–3921.

[41] I. Zarecki. (2024, October) Llm vector database: Why it's not enough for rag. K2view. Accessed: 2025-04-19. [Online]. Available: https://www.k2view.com/blog/llm-vector-database-why-its-not-enough-for-rag

[42] J. M. Zelle and R. J. Mooney, "Learning to parse database queries using inductive logic programming," in *Proceedings of the Thirteenth National Conference on Artificial Intelligence - Volume 2*, ser. AAAI'96. AAAI Press, 1996, p. 1050–1055.

[43] B. Zhang, Y. Ye, G. Du, X. Hu, Z. Li, S. Yang, C. H. Liu, R. Zhao, Z. Li, and H. Mao, "Benchmarking the text-to-sql capability of large language models: A comprehensive evaluation," 2024, arXiv:2403.02951 [cs.CL], version 2, 6 Mar 2024. [Online]. Available: https://arxiv.org/abs/2403.02951

[44] V. Zhong, C. Xiong, and R. Socher, "Seq2sql: Generating structured queries from natural language using reinforcement learning," 2017, arXiv:1709.00103 [cs.CL], version 7. [Online]. Available: https://doi.org/10.48550/arXiv.1709.00103

[45] Y. Zhu, J. R. A. Moniz, S. Bhargava, J. Lu, D. Piraviperumal, S. Li, Y. Zhang, H. Yu, and B.-H. Tseng, "Can large language models understand context?" https://github.com/apple/ml-llm-contextualization-eval, 2025, work conducted during internship at Apple. Accessed: 2025-04-25.

Appendix A

Supplemental Information

The following appendix presents detailed results and prompts for **Question 1** and **Question 2**, which does not contain sensitive information and can therefore be openly shared. **Question 3** and **Question 4** is some example questions. The customer has ben Anonymized. No SPARQL query will be given on these as that may compromise how the graph database is built and conflict with the NDA, this limited version still helps the reader to understand the questions even more.

Natural Language Question 1:
question 1: "Write a SPARQL query that lists all resources that are owl:Class, and include their rdfs:label if available."
Natural Language Question 2:
Question 2: Give me a SPARQL query that groups all instances by their RDF type (?instance a ?class) and counts how many instances each class has.
Natural Language Question 3:
Question 3: "Give me all resources that are part of *CompanyStrategyPrefix*:*CompanyStrategyObject* using the *is_part_of* predicate."
Natural Language Question 4:
Question 4: "Write a SPARQL query that returns all resources that are part of *CompanyStrategyPrefix*:*CompanyStrategyObject*, and for each one, include its title and description."

Correct SPARQL Query for Question 1:

```
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT ?class ?label WHERE {
  ?class a owl:Class .
  OPTIONAL { ?class rdfs:label ?label . }
}
```

Correct SPARQL Qurey for Question 2

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT ?class (COUNT(?instance) AS ?count)
WHERE {
```

```
  ?instance a ?class
}
GROUP BY ?class
```

A.1 Summary Metrics on question 1

The following table summarizes the performance metrics for the various approaches applied to Question 1:

Table A.1: Performance comparison for Question 1.

Method	Latency (s) (OAL)	Accuracy (AOEA)	Robustness	Total Tokens
Direct LLM (exp1)	3.396	1.000	1.00	22697
Vector RAG (exp2)	3.546	1.000	1.00	2157
ReAct (exp3)	3.522	1.000	1.00	10022
Multi-Agent (exp4)	10.383	0.143	1.00	1456
Langchain Chain (exp5)	5.670	1.000	0.86	16508

A.2 Summary Metrics

The following table summarizes the performance metrics for the various approaches applied to the questions:

Table A.2: Performance comparison for Question 1.

Method	Latency (s) (OAL)	Accuracy (AOEA)	Total Tokens	Robustness
Direct LLM	3.429	0.343	22668	0.83
Vector RAG	3.101	0.424	2511	0.73
ReAct	24.700	0.290	13703	0.70
Multi-Agent	11.513	0.310	1306	0.95
Langchain Chain	10.326	0.224	20811	0.98

A.3 Experimental Prompts

Experiment 1 (Direct LLM Prompting)

```
Generate a SPARQL query for the following question:
{question}

{SPARQL_PREFIXES}

Here is the ontology schema:

{ontology}

Answer ONLY with the SPARQL query, no other text or explanation etc just the query.
```

Experiment 2 (Retrieval-Augmented Generation - RAG)

```
I need to answer this question using SPARQL against a knowledge graph:
"{user_question}"

{SPARQL_PREFIXES}

Here is relevant context that might help formulate the query:

{context}

ANSWER WITH ONLY A SPARQL QUERY, DO NOT EXPLAIN THE QUERY.
```

Experiment 3 (Reasoning and Action Framework - ReAct)

```
System: You are a SPARQL-only reasoning agent, YOU MUST Answer the
question ONLY using SPARQL.

{SPARQL_PREFIXES}

Answer the following questions as best you can. You have access
to the following tools:

{{tools}}

Use the following format:

Question: the input question you must answer
Thought: you should always think about what to do next
Action: the action to take, should be one of [{{tool_names}}]
Action Input: the input to the action
Observation: the result of the action
... (this Thought/Action/Action Input/Observation can repeat N times)
Thought: I now know the final answer and have a SPARQL query that works
Final Answer: SPARQL

Begin!

Question: {{input}}
Thought:{{agent_scratchpad}}
```

Experiment 4 (Multi-Agent Architecture)

Retrieval Agent Prompt:

```
IMPORTANT: GIVE structured guidance and give all the needed classes,
object, subject or predicates for the question.
YOU CANNOT: ASK clarifying questions.
```

```
YOU CANNOT: give a SPARQL query.

{SPARQL_PREFIXES}

Thought:{{agent_scratchpad}}

The question: {{input}}
```

Generator Agent Prompt:

```
Your role is to craft a SPARQL query based on the user's natural
language question.
You have received the following context from the retrieval agent.
Use this context to generate a precise SPARQL query.

{SPARQL_PREFIXES}

RETRIEVER CONTEXT:
Generate a SPARQL query for this user question: {user_input}
```

Validator Agent Prompt:

```
You are a SPARQL validation assistant.
SPARQL_PREFIXES: {SPARQL_PREFIXES}

A test execution of this SPARQL query returned the following error:
{validation_result}

Original query:
{query}

Provide concise feedback to help a SPARQL Debugger agent fix the query,
focusing on syntax issues, missing prefixes, undefined variables, or invalid
classes/properties. Provide very concise, clear, and actionable feedback
with details.
```

Debugger Agent Prompt:

```
You are a SPARQL debugger agent.
SPARQL_PREFIXES: {SPARQL_PREFIXES}

A SPARQL query produced this error:
Original query that needs fixing:
{orig_query}

The error message is:
{error_msg}

Provide only the corrected SPARQL query as the output.
```

Faculty of Faculty, Blekinge Institute of Technology, 371 79 Karlskrona, Sweden