



UNIVERSITAS INDONESIA

FROG: FRAMEWORK OF OPEN GRAPH RAG

UNDERGRADUATE THESIS

**JAYCENT GUNAWAN ONGRIS / 2106750231 / COMPUTER SCIENCE
EDUARDUS TJITRAHARDJA / 2106653602 / COMPUTER SCIENCE**

**FACULTY OF COMPUTER SCIENCE
COMPUTER SCIENCE STUDY PROGRAM
DEPOK
DECEMBER 2024**



UNIVERSITAS INDONESIA

FROG: FRAMEWORK OF OPEN GRAPH RAG

UNDERGRADUATE THESIS

Submitted in partial fulfillment of the requirements for the degree
Sarjana Ilmu Komputer

JAYCENT GUNAWAN ONGRIS / 2106750231 / COMPUTER SCIENCE
EDUARDUS TJITRAHARDJA / 2106653602 / COMPUTER SCIENCE

FACULTY OF COMPUTER SCIENCE
COMPUTER SCIENCE STUDY PROGRAM
DEPOK
DECEMBER 2024

STATEMENT OF ORIGINALITY

**We hereby declare that this thesis is our work,
and all sources either cited or referred
have been properly acknowledged.**

Name of Author 1 : Jaycent Gunawan Ongris

NPM of Author 1 : 2106750231

Signature of Author 1 :



Name of Author 2 : Eduardus Tjitrahardja

NPM of Author 2 : 2106653602

Signature of Author 2 :



Date : 11 December 2024

HALAMAN PENGESAHAN

Tugas Akhir ini diajukan oleh :
Nama/NPM/Prodi Mahasiswa 1 : Eduardus Tjitrahardja/2106653602/Sarjana Ilmu Komputer
Nama/NPM/Prodi Mahasiswa 2 : Jaycent Gunawan Onggris/2106750231/Sarjana Ilmu Komputer
Judul : FrOG: Framework of Open GraphRAG

Telah berhasil dipertahankan di hadapan Dewan Penguji dan diterima sebagai bagian persyaratan yang diperlukan untuk memperoleh gelar Sarjana Ilmu Komputer pada Program Studi Sarjana Ilmu Komputer, Fakultas Ilmu Komputer, Universitas Indonesia

DEWAN PENGUJI

Pembimbing	: Dr.rer.nat. Fariz Darari, S.Kom., M.Sc.	(Nilai telah diberikan melalui SISIDANG pada 24-12-2024, 15:53:35) (Revisi telah disetujui melalui SISIDANG pada 05-01-2025, 16:46:37)
Pembimbing	: Dr.techn. Fajar J. Ekaputra, S.T., M.T.	(Nilai telah diberikan melalui SISIDANG pada 06-01-2025, 14:49:22) (Revisi telah disetujui melalui SISIDANG pada 06-01-2025, 14:49:55)
Penguji	: Amril Syalim, S.Kom., M.Eng., Ph.D.	(Nilai telah diberikan melalui SISIDANG pada 24-12-2024, 15:30:00) (Revisi telah disetujui melalui SISIDANG pada 06-01-2025, 14:38:12)
Penguji	: Muhammad Febrian Rachmadi, S.Kom., M.Sc., Ph.D.	(Nilai telah diberikan melalui SISIDANG pada 24-12-2024, 15:47:15) (Revisi telah disetujui melalui SISIDANG pada 03-01-2025, 10:18:30)

Ditetapkan di : Depok, Jawa Barat
Tanggal : 06 Januari 2025

PREFACE

First and foremost, we express our highest praise and gratitude to the Almighty, whose blessings have enabled us to complete our journey in Fasilkom UI, and specifically, this undergraduate thesis entitled “FrOG: Framework of Open GraphRAG”. We would also like to express our deepest gratitude to those who have supported and guided us throughout this process:

- Our sincere thanks go to our supervisors, Dr.rer.nat. Fariz Darari, S.Kom., M.Sc. and Dr.techn. Fajar Juang Ekaputra, S.T., M.T., for their invaluable guidance, patience, and expertise. Their insightful feedback and constant encouragement were essential in helping us navigate the challenges of this research.
- We are also grateful to all the professors and lecturers at Fasilkom UI, whose knowledge and teachings have shaped our academic journey and contributed significantly to the completion of this thesis.
- We are deeply grateful to Universitas Indonesia for providing an inspiring and challenging academic environment. The resources, faculty, and opportunities at the university have greatly supported our development throughout our studies.
- Our heartfelt thanks to Wikimedia Indonesia for their generous support through the Wikidata Research Grant 2024, which made this research possible. Their contribution played a key role in the development of this thesis and research, in general.
- To our beloved parents, thank you for your endless love, sacrifices, and belief in us. Your constant support has been a source of strength throughout this entire process.
- We would like to express our gratitude to our dear friends for their companionship, understanding, and encouragement. Their moral support and shared moments of joy made this challenging journey more bearable.

We acknowledge that this thesis is far from perfect, and any mistakes or shortcomings are our responsibility. Therefore, we welcome any constructive feedback or suggestions for improvement. Please feel free to contact us via email at jaycentgunawanong@gmail.com and edutjitrahardja@gmail.com.

Finally, we express our profound appreciation to everyone who has contributed to this work. Your support has made this accomplishment possible.

Depok, 11 December 2024

Authors

STATEMENT OF CONSENT OF THESIS PUBLICATION FOR ACADEMIC INTERESTS

As members of Universitas Indonesia's academic community, we the undersigned below:

Author 1

Name : Jaycent Gunawan Ongris
NPM : 2106750231
Study Program : Computer Science

Author 2

Name : Eduardus Tjitrahardja
NPM : 2106653602
Study Program : Computer Science
Academic Work : Undergraduate Thesis

in the scientific development interest, hereby consent to grant Universitas Indonesia a **Non-exclusive Royalty Free Right** of scientific work entitled:

FrOG: Framework of Open GraphRAG

along with existing sets of supplementary materials (if required). With Non-exclusive Royalty Free Right, Universitas Indonesia has rights to save, media-transfer/format, manage in a database, maintain, and publish my thesis provided our names are retained as the authors/creators and the owners of Copyright.

We certify that the statement is true to the best of our knowledge.

Ratified in : Depok

Date : 11 December 2024

Signed,

Author 1:

A handwritten signature in black ink, consisting of a large, stylized 'J' followed by a series of loops and a horizontal line at the bottom.

(Jaycent Gunawan Onggris)

Author 2:

A handwritten signature in black ink, featuring a large, stylized 'E' followed by a series of loops and a horizontal line at the bottom.

(Eduardus Tjitrahardja)

ABSTRACT

Author 1 / Study Program : Jaycent Gunawan Ongris / Computer Science
Author 2 / Study Program : Eduardus Tjitrahardja / Computer Science
Title : FrOG: Framework of Open GraphRAG
Advisors : Dr.rer.nat. Fariz Darari, S.Kom., M.Sc.
Dr.techn. Fajar Juang Ekaputra, S.T., M.T.

The rise of large language models (LLMs) has advanced information retrieval, but issues like limited knowledge updating, lack of openness, and hallucinations persist. Retrieval-Augmented Generation (RAG) addresses these, though it lacks interpretability due to reliance on vector-based representations. Our research presents a RAG framework using a knowledge graph (KG) as the primary knowledge base, with only open-source components to allow for user customization. The main pipeline consists of entity linking, retrieval using verbalized sentences or SPARQL query generation, and answer generation, integrated with ontology (properties and classes) retrieval via a vector database. The pipeline is tested on Wikidata, DBpedia, and a local domain-specific KG, achieving accuracies of 0.458, 0.517, and 0.805, respectively. An ablation study further reveals that ontology retrieval is the most critical component in providing context to the LLM for generating accurate SPARQL queries.

Key words:

LLM, RAG, KG, GraphRAG, Wikidata, DBpedia, SPARQL

ABSTRAK

Nama Penulis 1 / Program Studi : Jaycent Gunawan Ongris / Ilmu Komputer
Nama Penulis 2 / Program Studi : Eduardus Tjitrahardja / Ilmu Komputer
Judul : FrOG: Framework of Open GraphRAG
Pembimbing : Dr.rer.nat. Fariz Darari, S.Kom., M.Sc.
Dr.techn. Fajar Juang Ekaputra, S.T., M.T.

Munculnya *large language models* (LLMs) telah memajukan bidang *information retrieval*, tetapi masalah seperti keterbatasan pembaruan pengetahuan, kurangnya keterbukaan, dan halusinasi masih ada. *Retrieval-Augmented Generation* (RAG) membantu mengatasi beberapa tantangan ini tetapi memiliki keterbatasan dalam interpretabilitas karena bergantung pada representasi berbasis vektor. Penelitian ini memperkenalkan sebuah kerangka kerja RAG yang menggunakan graf pengetahuan (KG) sebagai basis pengetahuan utama, dengan hanya menggunakan komponen *open-source* untuk memungkinkan kustomisasi pengguna. *Pipeline* utama terdiri dari penghubungan entitas, *retrieval* menggunakan kalimat hasil verbalisasi atau pembuatan kueri SPARQL, dan pembangkitan jawaban, yang terintegrasi dengan *retrieval* ontologi (properti dan kelas) melalui basis data vektor. Kerangka kerja ini diuji pada Wikidata, DBpedia, dan KG dengan domain spesifik lokal, dengan akurasi masing-masing 0,458, 0,517, dan 0,805. Studi ablatasi lebih lanjut mengungkapkan bahwa *retrieval* ontologi adalah komponen yang paling utama dalam memberikan konteks kepada LLM untuk bisa membangkitkan kueri SPARQL yang akurat.

Kata kunci:

LLM, RAG, KG, GraphRAG, Wikidata, DBpedia, SPARQL

TABLE OF CONTENTS

TITLE PAGE	i
STATEMENT OF ORIGINALITY	ii
LEMBAR PENGESAHAN	iii
PREFACE	iv
STATEMENT OF CONSENT OF THESIS PUBLICATION	vi
ABSTRACT	viii
TABLE OF CONTENTS	x
LIST OF FIGURES	xiii
LIST OF TABLES	xiv
LIST OF CODES	xiv
LIST OF APPENDIX	xv
 1 INTRODUCTION	 1
1.1 Overview	1
1.2 Research Questions	3
1.3 Research Objectives	4
1.4 Research Scopes	4
1.5 Research Stages	5
1.6 Outline	6
 2 LITERATURE REVIEW	 7
2.1 Semantic Web and Knowledge Graphs	7
2.1.1 Identifier: Uniform Resource Identifier (URI)	8
2.1.2 Character Set: Unicode	9
2.1.3 Syntax	9
2.1.3.1 RDF/XML	9
2.1.3.2 JSON-LD	10
2.1.3.3 Turtle	10
2.1.4 Data Interchange: Resource Description Framework (RDF)	11
2.1.4.1 RDF/XML	13
2.1.4.2 JSON-LD	14
2.1.4.3 Turtle	15
2.1.5 Taxonomy: RDF Schema (RDFS)	15
2.1.6 Rule: Rule Interchange Format (RIF)	16
2.1.7 Ontology: Web Ontology Language (OWL)	17
2.1.8 Querying: SPARQL	17
2.1.9 Other Components	19
2.2 Large Language Model (LLM)	19
2.2.1 General Architecture of Transformer	20
2.2.1.1 Encoder	21
2.2.1.2 Decoder	22
2.2.1.3 Attention	22
2.2.2 Example Models	24

2.2.2.1	BERT (Bidirectional Encoder Representations for Transformers)	24
2.2.2.2	T5 (Text-To-Text Transfer Transformer)	25
2.2.2.3	GPT (Generative Pre-trained Transformer)	26
2.2.2.4	MPNet	28
2.2.2.5	LLaMA 3.1	29
2.2.2.6	Jina Embeddings v3	29
2.2.2.7	Mistral 7B	31
2.2.2.8	Mistral NeMo	31
2.2.2.9	Qwen2.5	32
2.2.2.10	Qwen2.5-Coder	32
2.3	Retrieval-Augmented Generation (RAG)	33
2.3.1	RAG vs. Fine-tuning	34
2.3.2	Components	35
2.3.2.1	Retriever	36
2.3.2.2	Generator	37
2.3.3	GraphRAG	37
2.4	Comparison of Several QA Systems based on KG or LLM	39
2.4.1	KG-based QA Systems	40
2.4.2	LLM-based QA Systems	40
2.4.3	Text-based RAG QA Systems	41
2.4.4	GraphRAG QA Systems	42
2.4.5	Performance and Accuracy Comparison	43
3	METHODOLOGIES	47
3.1	Research Methodology	47
3.2	Experiment Details	48
3.3	Knowledge Bases	49
3.3.1	Wikidata	49
3.3.2	DBpedia	51
3.3.3	Local KGs	53
3.4	Ontology	56
3.4.1	Wikidata	56
3.4.2	DBpedia	56
3.4.3	Local KGs	56
3.5	Dataset	57
3.5.1	Pipeline v1 Dataset	57
3.5.2	Pipeline v2 Dataset	58
3.5.2.1	Wikidata and DBpedia	58
3.5.2.2	Local KGs	60
3.6	Evaluation Metric	64
4	IMPLEMENTATION	67
4.1	Pipeline v1	67
4.1.1	Entities Extraction	67
4.1.2	Entity ID Retrieval	69
4.1.3	Entity Disambiguation and Selection	69

4.1.4	SPARQL Query Generation	71
4.1.5	Query Execution and Answer Generation	74
4.2	Pipeline v2	75
4.2.1	Translation	76
4.2.2	Entities Linking	77
4.2.3	Retrieval	79
4.2.3.1	Simple Question	79
4.2.3.2	Complex Question	81
4.2.4	Answer Generation	85
4.3	Libraries and Resources	86
4.4	Computational Setup	88
5	RESULT AND ANALYSIS	89
5.1	Experiment Results	89
5.1.1	Dataset Pipeline v1	89
5.1.2	Dataset Pipeline v2	91
5.1.2.1	Wikidata	91
5.1.2.2	DBpedia	94
5.1.2.3	Local KGs	96
5.1.2.4	Multilingual Support	97
5.1.3	General Analysis	98
5.1.3.1	LLMs	98
5.1.3.2	Knowledge Bases	99
5.2	Ablation Study	100
5.2.1	Verbalization	101
5.2.2	Chain of Thoughts (CoT)	103
5.2.3	Few Shots	104
5.2.4	Entity and Property (Ontology) Retrieval from Vector Database . .	106
5.2.5	Conclusion	107
5.3	Dataset Issues	108
6	CONCLUSION	110
6.1	Conclusion	110
6.2	Future Works	111
	BIBLIOGRAPHY	113

LIST OF FIGURES

Figure 1.1.	The Evolution of QA Systems	3
Figure 2.1.	Semantic Web Layer Cake (Obitko, 2007)	8
Figure 2.2.	RDF Graph Example (Vanderbilt Libraries, 2019)	12
Figure 2.3.	RDF/XML for Triples in Table 2.1	13
Figure 2.4.	JSON-LD for Triples in Table 2.1	14
Figure 2.5.	JSON-LD with <code>@context</code> for Triples in Table 2.1	15
Figure 2.6.	Turtle for Triples in Table 2.1	15
Figure 2.7.	SQL and SPARQL Syntax Comparison	18
Figure 2.8.	The Transformer’s Architecture (Vaswani et al., 2017)	21
Figure 2.9.	Scaled Dot-Product Attention and Multi-Head Attention (Vaswani et al., 2017)	23
Figure 2.10.	Pre-Training and Fine-Tuning in BERT (Devlin et al., 2018)	25
Figure 2.11.	Usages of T5 (Raffel et al., 2019)	26
Figure 2.12.	BERT vs. GPT (Raschka, 2024)	27
Figure 2.13.	Jina Embeddings v3 Model Architecture (Sturua et al., 2024)	30
Figure 2.14.	Qwen2.5-Coder Training Pipeline (Hui et al., 2024)	33
Figure 2.15.	RAG Approach Overview (Lewis et al., 2021)	35
Figure 2.16.	Query-based GraphRAG vs. Naïve Text-based RAG	39
Figure 3.1.	Wikidata Query Service’s Interface	50
Figure 3.2.	Wikidata Page of “instance of” (<code>wdt:P31</code>)	51
Figure 3.3.	Wikidata Page of “house cat” (<code>wd:Q146</code>)	51
Figure 3.4.	DBpedia’s SPARQL Query Editor Interface	52
Figure 3.5.	DBpedia Page of “birth date” (<code>dbo:birthDate</code>)	53
Figure 3.6.	DBpedia Page of “Lionel Messi” (<code>dbr:Lionel_Messi</code>)	53
Figure 3.7.	Evaluation Schema	66
Figure 4.1.	The Initial Pipeline	67
Figure 4.2.	The Upgraded Pipeline	75
Figure 4.3.	Template-Based Verbalization Overview	81

LIST OF TABLES

Table 2.1.	RDF Triples about Julian Hillyer (Vanderbilt Libraries, 2019)	12
Table 2.2.	Namespace Prefixes and Corresponding URIs (Vanderbilt Libraries, 2019)	13
Table 2.3.	RDFS Syntax Examples (Allemang and Hendler, 2011)	16
Table 3.1.	Properties Mapping	55
Table 3.2.	Ontology of the Local Curriculum KG	57
Table 3.3.	Sample Queries from Dataset v1	58
Table 3.4.	Sample Queries from Dataset v2 for Wikidata and DBpedia (QALD) .	60
Table 3.5.	Supported Question Categories	61
Table 3.6.	Sample Questions for the Local Curriculum KG	63
Table 4.1.	Several Few-Shot Examples of Entity Extraction	79
Table 4.2.	Several Few-Shot Examples of Related Properties Generation (DBpedia)	82
Table 5.1.	Experiment Results on the First Pipeline’s Dataset	89
Table 5.2.	Experiment Results on the Downsampled Wikidata QALD-9-Plus Dataset	91
Table 5.3.	Experiment Results on the Downsampled DBpedia QALD-9-Plus Dataset	94
Table 5.4.	Experiment Results on the Local Curriculum KG Dataset	96
Table 5.5.	Experiment Results on the Downsampled Wikidata QALD-9-Plus Dataset Indonesian Version	97
Table 5.6.	Ablation Study Results on Verbalization	101
Table 5.7.	Ablation Study Results on CoT	103
Table 5.8.	Ablation Study Results on Few Shots	104
Table 5.9.	Ablation Study Results on Ontology Retrieval	106

LIST OF CODES

Code 3.1.	Triples about “Basis Data” Course	54
Code 3.2.	RDF Version of Code 3.1	55
Code 3.3.	Python Code for Dataset Downsampling	59
Code 4.1.	Equivalent SPARQL Query to Retrieval in Figure 4.3	81
Code 4.2.	Several Few-Shot Examples of SPARQL Generation (DBpedia)	84
Code 4.3.	Wikidata SPARQL Query to Obtain Labels	85
Code 4.4.	DBpedia (or other KGs) SPARQL Query to Obtain Labels	86
Code 5.1.	Ground Truth of “location name of the Mona Lisa”	90
Code 5.2.	Prediction of “location name of the Mona Lisa”	91
Code 1.	Code for Local KG Dataset Generator	127
Code 2.	JSON for QALD Test Dataset	134

LIST OF APPENDIX

Appendix 1. Local KG Dataset Generator	127
Appendix 2. QALD Test Dataset	134
Appendix 3. Task Distribution	140
Appendix 4. FrOG Github README	140

CHAPTER 1

INTRODUCTION

In this chapter, we will explain the background motivation and problems we aim to resolve through this research.

1.1 Overview

Recent advancements in the field of large language models (LLMs) have transformed the paradigm of Information Retrieval (IR) applications (Dai et al., 2024). This phenomenon can be seen nowadays, where numerous LLM-powered services like OpenAI's ChatGPT, Google's Gemini, and Microsoft's Copilot are employed for IR tasks, such as question-answering (QA) and text summarization. Specifically, for search engines, existing works focus on leveraging LLMs to improve query rewriting, retrieval, reranking, and content reading (Zhu et al., 2024).

LLMs can contribute significantly to the field of IR due to their impressive ability to understand natural language semantically and generate natural language text fluently. These skills are acquired during the pre-training phase, where massive amounts of textual data are fed into models to optimize their parameters. The knowledge learned during this phase is embedded within the models' parameters, making such models known as *parametric models*. These foundation models can be further fine-tuned with instruction datasets or specific examples to perform many different downstream tasks (Ziegler et al., 2020). Additionally, they can also learn from human feedback and in-context learning, i.e., the model uses natural language instructions stated in the prompt as a guidance to complete similar tasks by simply predicting which token will come up next (Brown et al., 2020).

Despite their performance, there are still several limitations of LLMs: the difficulties in expanding or updating the knowledge, the lack of transparency or interpretability (or the *black-box* problem of AI), and the “hallucination” phenomena (Lewis et al., 2021), where the generated response is plausible but nonfactual. These limitations—especially the last one—have led to significant concerns about the trustworthiness of LLMs. Consequently, people tend to be reluctant when it comes to leveraging LLMs for decision-making in

critical fields like law, healthcare, finance, and other sectors, where accountability and accuracy are the most important aspects.

To get rid of these problems while still maintaining the excellent performance of LLMs, *retrieval-augmented generation* (RAG) was introduced (Lewis et al., 2021). This approach does not rely solely on parametric models but also incorporates external *non-parametric models* which perform retrieval on a knowledge base. By using this component, the knowledge can be easily expanded or updated as it is not embedded in the parameters. Moreover, this component also allows direct inspection of the retrieved data, partially addressing the black-box problem. However, while this approach does not guarantee that hallucinations will disappear completely, it reduces them by not depending solely on the knowledge embedded within the LLMs.

The original naïve RAG approach as proposed by Lewis et al. (2021) uses Dense Passage Retriever (DPR) as the non-parametric component (Karpukhin et al., 2020). This method encodes both query and documents into dense embeddings, such as using BERT_{BASE} (Lewis et al., 2021). The maximum inner product search (MIPS) algorithm is then employed to find the top- K most relevant documents given the query. However, despite DPR outperforming traditional scoring algorithms like BM25, it still suffers from the black-box problem due to the nature of dense embeddings which are generally less interpretable. Furthermore, the process of splitting large documents into smaller ones for encoding (i.e., chunking) can potentially lead to the loss of contextual information and disrupt the coherence of the original content if related information is split across different chunks (Sarmah et al., 2024).

Therefore, to address those issues, one approach is to use a different knowledge base to store the knowledge to be retrieved, instead of a corpus of text chunks. One of the most versatile knowledge bases is the knowledge graph (KG), which stores information in the form of a graph structure. Leveraging KG as the knowledge base offers several advantages: the entities in the KG which are contextualized through their connections to other entities within the KG that can provide meaningful semantic relationships, the graph structure that supports graph theory algorithms (e.g., community detection) to extract meaningful information, and the ability to provide reasoning over retrieval results through graph walks (i.e., tracing the paths connecting one entity to another) (Abu-Rasheed et al.,

2022). To sum up all the advancements in QA systems as explained before, we provide Figure 1.1.

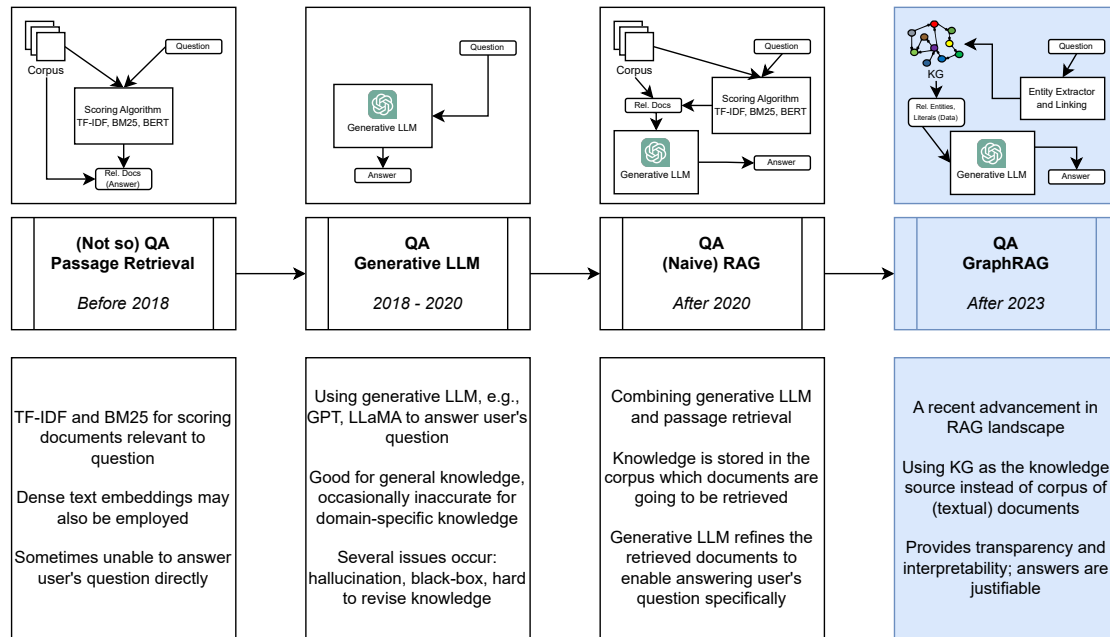


Figure 1.1: The Evolution of QA Systems

Considering the lack of interpretability and transparency problems exhibited by most LLMs and naïve RAG, we propose **FrOG: Framework of Open GraphRAG**. FrOG is a RAG system that uses a KG as the primary knowledge base, which relies solely on open-source components. This approach offers direct tracing of the retrieved knowledge, making the system's reasoning process more transparent and interpretable. Additionally, we are using off-the-shelf open-source LLMs, allowing users to implement the system directly without fine-tuning, which process requires significant computational resources. By building the system with open-source components, we also promote accessibility and flexibility, making it easier to tailor the system for domain-specific applications.

1.2 Research Questions

The main goal of this research is to design and implement a KG-agnostic RAG system composed of only open-source components. To guide our investigation, we formulate our research questions below.

1. How to implement RAG using KG as the primary knowledge base, leveraging only

open-source components?

2. Which open-source LLM that serves as the basis for this GraphRAG architecture and achieves the highest answer accuracy?
3. Which specific component within the architecture plays the most crucial role in ensuring the generation of accurate answers?

1.3 Research Objectives

Based on the aforementioned research questions, the research objectives we aim to achieve are as follows.

1. To develop a RAG system using KG as the primary knowledge base, leveraging only open-source components.
2. To identify open-source LLM that serves as the basis for this GraphRAG architecture and achieves the highest answer accuracy.
3. To determine the most influential components within the architecture that contribute to the accuracy of generated answers.

1.4 Research Scopes

To ensure a more focused and manageable research, we limit the research scopes as follows.

1. The proposed system is not designed to handle descriptive questions. The system focuses on factoid questions, which are questions with a single and concise answer, such as specific entities, dates, or numerical values, e.g., “Who is the richest person in the world?”. Additionally, listing questions are also supported, e.g., “Mention all books written by Shakespeare.”.
2. We evaluate the system on five generative LLMs: LLaMA 3.1 8B, Mistral 7B, Mistral NeMo, Qwen 2.5 7B, and Qwen 2.5 7B Coder. These models were chosen for their popularity within the AI community, their open-source nature, and their alignment with our system’s requirements. Specifically, we focus on models within the 7B-12B parameter range due to the feasibility of implementation.
3. Our system implementation supports three knowledge bases: Wikidata, DBpedia, and a locally developed curriculum-related KG. Wikidata and DBpedia are selected as representatives of open KGs, while the curriculum KG serves as an example of an

enterprise KG.

4. For DBpedia, our proposed system can only handle questions incorporating `http://dbpedia.org/ontology/` or `:dbo` prefix. This limitation arises because the system relies on the T-Box ontology, which defines the schema and vocabulary of DBpedia for the `:dbo` namespace only.
5. For the local knowledge graph, our current implementation uses a curriculum-related knowledge graph, which is relatively compact, consisting of 874 triples. The underlying implementation leverages `rdflib` to store the graph in memory, showcasing its capability to handle substantial graph sizes effectively, depending on the available system memory. For example, on an Ubuntu 12.04 amd64 system with 8 GB of RAM, `rdflib` has been shown to load up to 15.7M triples (from a 738MB Turtle file) while using approximately 4.8 GB of memory¹.

1.5 Research Stages

The research is carried out in several stages, as outlined below.

1. Problem Definition

This stage focuses on formulating the research objectives and identifying the key problems the study aims to address.

2. Literature Review

In this stage, the necessary background knowledge for the research is gathered. Topics of focus include Knowledge Graphs (KGs), Large Language Models (LLMs), Retrieval-Augmented Generation (RAG), and GraphRAG.

3. Framework Design

The design of the GraphRAG framework takes place in this stage, ensuring alignment with the research objectives.

4. Framework Implementation

Here, the designed framework is implemented into a functional system, adhering to the established specifications.

5. Experimentation and Evaluation

Several experiments are conducted to evaluate the performance of the implemented system, providing insights into its effectiveness and potential limitations.

¹<https://groups.google.com/g/rdflib-dev/c/EZqiUs7qTUc?pli=1>

6. Conclusion

This final stage summarizes the research outcomes based on the evaluation results. It also explores possible directions for future work and improvements.

1.6 Outline

This report is organized as follows.

- Chapter 1 INTRODUCTION

This chapter introduces the research by providing the background for the study, problem definition (i.e., research questions, research objectives, and research scopes), and research stages.

- Chapter 2 LITERATURE REVIEW

This chapter elaborates on the theoretical foundations and prior research related to the study. It discusses the topics on the KG, LLM, RAG, and GraphRAG.

- Chapter 3 METHODOLOGIES

This chapter outlines the research methodologies we employ in this study, including the general framework, knowledge bases, datasets, and evaluation metrics.

- Chapter 4 IMPLEMENTATION

This chapter details the practical implementation of the research, including the specific pipeline explanation and the libraries utilized to construct the system.

- Chapter 5 RESULT AND ANALYSIS

This chapter presents the outcomes of the research, including the results of system testing and a detailed analysis of the findings. Furthermore, it includes an ablation study to evaluate the contributions of individual components of the system.

- Chapter 6 CONCLUSION

This chapter summarizes the research findings and their implications. Additionally, it also provides recommendations for future research and development.

CHAPTER 2

LITERATURE REVIEW

In this chapter, we will explain several theoretical foundations and prior research works related to our work.

2.1 Semantic Web and Knowledge Graphs

The concept of knowledge graphs originates from the notion of the Semantic Web. The Semantic Web itself is considered as an extension of the Web, aiming to make it understandable by computers, not only humans. To achieve this behavior, large collections of structured information and sets of inference rules are needed by computers to perform reasoning (Berners-Lee et al., 2001). This structured approach enables computers to interpret and derive meaningful insights from a variety of data sources.

On the other hand, knowledge graphs take the principles of the Semantic Web further by putting them into practical use. Introduced by Google in 2012, knowledge graphs represent information in the form of a graph, where entities are depicted as nodes and semantic relationships between entities are depicted as edges. With knowledge graphs, information retrieval results can be enhanced significantly by capturing the context and entities related to the query, not just relying on conventional text-matching techniques (Singhal, 2012). Furthermore, knowledge graphs also can be utilized to increase the explainability of retrieval algorithms, as part of *explainable information retrieval* (XIR), since most state-of-the-art vector-based algorithms fail to explain why a certain document is retrieved as a result of a certain query (Chen et al., 2023).

Leveraging structured data, the Semantic Web (and knowledge graph, in general) require a standardized format to describe the entities and relationships. This format forms the backbone of the Semantic Web's general architecture, which is known as the Semantic Web Layer Cake, as displayed in Figure 2.1. The Semantic Web Layer Cake illustrates the hierarchical structure of the Semantic Web, where each layer utilizes the capabilities of the layers below.

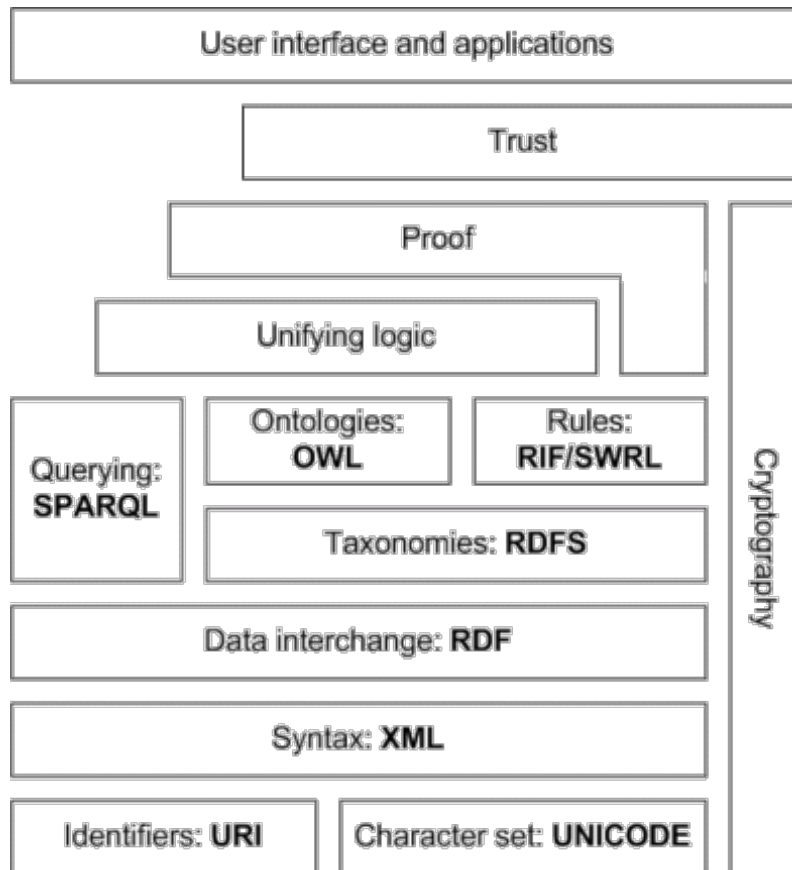


Figure 2.1: Semantic Web Layer Cake (Obitko, 2007)

Components of the Semantic Web, as visualized in Figure 2.1, will be described in detail in the subsequent subsubchapters.

2.1.1 Identifier: Uniform Resource Identifier (URI)

As the Semantic Web adopts the structured approach, standardized identifiers are needed to uniquely distinguish items in the Semantic Web. This is because the name of the item itself is not adequate to differentiate between different items, as they may have the same name. For instance, the term “Taj Mahal” can refer to two different things: the famous mausoleum in India and the American Blues musician. If those items are represented in the Semantic Web, they will be mapped to different URIs.

The concept of URI is actually a generalization of the Uniform Resource Locator (URL), which is used to identify web pages in the Web. The Semantic Web works at the level of the data rather than at the level of the presentation (Allemang and Hendler, 2011), i.e., the web page. Hence, instead of having URLs to refer to web pages like the Web, the

Semantic Web uses URIs to refer to data items (entities and relationships).

2.1.2 Character Set: Unicode

Unicode is a standardized text encoding scheme maintained by The Unicode Consortium. The scheme supports all international character sets, therefore, making all human languages can be represented using it. Moreover, it also encodes various kinds of symbols, dingbats, and even emojis. As of June 14th, 2024, the latest version of Unicode is 15.1.0, which supports a total of 149,813 characters (The Unicode Consortium, 2023). In the realm of the Semantic Web, the usage of Unicode ensures that the information is encoded in a consistent and universally-understood format.

2.1.3 Syntax

Within the context of the Semantic Web, there are several standardized syntaxes to represent information in the form of Resource Description Framework (RDF): RDF/XML, Turtle, N-Triples, RDF-a, and JSON-LD (Harth et al., 2014). Originally, the World Wide Web Consortium (W3C) recommends the use of RDF/XML for RDF serialization, since the Web infrastructures are accustomed to representing information in HTML (Allemang and Hendler, 2011), which is the specific form of XML. However, since JSON is the most popular way to serialize data in web applications nowadays, it would be relatively easy for many programmers to interpret the format of JSON-LD. In terms of human readability, the Turtle serialization stands out, thanks to its intuitive representation of triples, by simply listing three resources in subject/predicate/object order using QName abbreviations¹, followed by a period (Allemang and Hendler, 2011). Each syntax will be explained subsequently.

2.1.3.1 RDF/XML

RDF/XML utilizes the XML syntax in representing RDF data. Just like HTML, XML is a markup language which mainly used for storing data in structured format, i.e. the tag format. However, in XML, users are allowed to define their own tags based on the data characteristics they want to work on (Decker et al., 2000). To give a context, an XML document consists of tags, with each tag is denoted by a pair of angle brackets, with text

¹ Short representations of URIs, will be explained in 2.1.3.3.

between them to indicate the name of the element which the tag creates. Furthermore, other XML elements can be placed as the contents of an XML element.

Despite being the most well-known and widely-supported RDF serialization due to its XML structure, there are several downsides on the usages of RDF/XML. The main problem lies within the representation itself: the mixture of a tree-like document and a triple-based graph (Meindertma, 2019), which presents an awkward representation of the RDF data. It does not explicitly reflect the triple model, so it might be quite difficult and cumbersome for people to interpret RDF/XML compared to other serialization methods.

2.1.3.2 JSON-LD

As stated in the name, the JSON-LD uses JSON structure to represent RDF data. JSON (JavaScript Object Notation) is a standardized, language-independent data interchange format which stores data in the form of two structures: a collection of name-value pairs and an ordered list of values (JSON, nd). It is used commonly in web applications to transmit data (Mozilla Developer Network, nd) due to its simple structure and intuitiveness.

JSON-LD itself is an extension of the plain JSON. It is easier to read and understand JSON-LD compared to RDF/XML, making it as one of the most common RDF representations nowadays. To justify this, according to surveys by W3Techs (2024), as of July 13th, 2024, JSON-LD is used by approximately 48.6% of all the websites, including the ones established by big companies, such as Google, Microsoft, Apple, YouTube, and many more. Nevertheless, the disadvantage of using JSON-LD is that it is difficult and costly to parse, especially when the RDF data is needed instead of the JSON object, since the parsing often involves requesting data from the internet (Meindertma, 2019).

2.1.3.3 Turtle

Besides JSON-LD, Turtle (the Terse RDF Triple Language) is considerably one of the most popular RDF serialization techniques nowadays. Turtle is known for its simple, concise, and highly readable syntax. It is the subset of Notation3 (N3) serialization and also a much simpler version of it. Consequently, parsing Turtle is way simpler than parsing N3, albeit it is more costly to parse Turtle compared to N-Triples (Meindertma, 2019). Despite this, Turtle serialization is widely used, including the usage within the

Semantic Web architecture itself. For instance, in SPARQL, the syntax of Turtle is used to define the query pattern for data retrieval.

Turtle representation usually starts with the prefix declaration, which denoted by `@prefix`. Within this section, bindings between namespace prefixes and global URIs are defined. Another commonly used term is `QName`, which is the short and compact representation of URI that combines a namespace prefix and a local name (identifier) for the resource. For instance, the namespace prefix `rdf` is associated with the URI `http://www.w3.org/1999/02/22-rdf-syntax-ns#`, which is declared within `@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .` On the other hand, whenever a `QName` like `rdf:type` is called, it refers to the URI `http://www.w3.org/1999/02/22-rdf-syntax-ns#type`, constructed by appending the local name `type` to the namespace URI.

Once the namespace prefixes have been defined, the triples can be easily expressed by listing three resources using `QName` abbreviations in subject/predicate/object order, followed by a period (Allemang and Hendler, 2011). Furthermore, Turtle also provides concise representation for several triples which share the same subject to avoid verbosity. In this representation, the subject is only specified once in the first triple; the rest of the triples only need to specify the predicates and objects. However, there is a slight difference in terms of syntax. Instead of terminating each triple that shares the same subject with a period, it uses a semicolon. Only the latter one uses a period as the terminating symbol.

2.1.4 Data Interchange: Resource Description Framework (RDF)

Resource Description Framework (RDF) is the foundation of the Semantic Web. It provides a standardized model for data interchange in the Semantic Web (DeWeese and Segal, 2014). As expressed by the name, RDF is a framework to describe information about resources. All things in the world can be referred to as *resources* in the Semantic Web (Allemang and Hendler, 2011). This term is actually synonymous with *entity*—introduced earlier—although according to Allemang and Hendler (2011), despite its intuitiveness, the usages of *entity* have certain issues.

The main component of RDF is called the *triple*. Following the notion of the elementary

linguistics grammar, a triple consists of a subject, a predicate, and an object. A subject is an information resource represented by its URI, a predicate describes the relationship between the subject and the object, and it is also represented by its URI, while an object can be a value or a URI to another resource (DeWeese and Segal, 2014). To put it simply, an object is what a predicate of a subject describes.

It is very common to represent RDF datasets—which consist of triples—as directed labeled graphs, with subjects and objects depicted as vertices and predicates depicted as edges (Harth et al., 2014). This representation is useful when dealing with multiple triples that refer to a single resource, as it eases interpretation. Furthermore, it allows data from multiple sources to be merged easily by simply forming a graph composed by all triples from these sources, i.e., individual graphs (Allemang and Hendler, 2011).

An example of simple RDF graph about an entity that represents Julian Hillyer can be seen in Figure 2.2 with the triples in Table 2.1. Due to long representations, the resources are represented using QNames, which namespace prefixes are defined in Table 2.2.



Figure 2.2: RDF Graph Example (Vanderbilt Libraries, 2019)

Subject	Predicate	Object
orcid:0000-0002-3178-0201	dcterms:created	2014-12-22T22:25:56.900Z
orcid:0000-0002-3178-0201	mads:hasAffiliation	http://www.grid.ac/institutes/grid.152326.1
orcid:0000-0002-3178-0201	rdf:type	foaf:Person
orcid:0000-0002-3178-0201	rdfs:label	“Julian Hillyer”

Table 2.1: RDF Triples about Julian Hillyer (Vanderbilt Libraries, 2019)

Namespace Prefix	URI
rdf	http://www.w3.org/1999/02/22-rdf-syntax-ns#
rdfs	http://www.w3.org/2000/01/rdf-schema#
xsd	http://www.w3.org/2001/XMLSchema#
foaf	http://xmlns.com/foaf/0.1/
dcterms	http://purl.org/dc/terms/
orcid	http://orcid.org/
mads	http://www.loc.gov/mads/rdf/v1#

Table 2.2: Namespace Prefixes and Corresponding URIs (Vanderbilt Libraries, 2019)

Representations of these triples in RDF/XML, JSON-LD, and Turtle will be explained subsequently.

2.1.4.1 RDF/XML

The RDF/XML representation of the triples in Table 2.1 can be seen in Figure 2.3.

```

1 <?xml version="1.0" encoding="utf-8" ?>
2 <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
3   xmlns:foaf="http://xmlns.com/foaf/0.1/"
4   xmlns:dcterms="http://purl.org/dc/terms/"
5   xmlns:mads="http://www.loc.gov/mads/rdf/v1#"
6   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
7
8   <foaf:Person rdf:about="http://orcid.org/0000-0002-3178-0201">
9     <dcterms:created rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">2014-12-22T22:25:56.900Z</dcterms:created>
10    <mads:hasAffiliation rdf:resource="http://www.grid.ac/institutes/grid.152326.1"/>
11    <rdfs:label>Julian Hillyer</rdfs:label>
12  </foaf:Person>
13 </rdf:RDF>

```

Figure 2.3: RDF/XML for Triples in Table 2.1

At the top of the RDF/XML document, the version and encoding scheme of the XML are stated within the `xml` tag. The triple data are defined within the `rdf:RDF` element, which acts as the root element. The namespace declaration is placed inside the opening `<rdf:RDF>` tag, where namespace prefixes are specified as attributes with their URIs as the values. Triples about a certain entity, i.e., Julian Hillyer are put inside an element. In this case, the element refers to an instance of `Person` class whose ID is 0000-0002-3178-0201. This element acts as the subject. Other predicate and object pairs that relate to the `Person` instance as the subject, i.e. `dcterms:created`, `mads:hasAffiliation`, and `rdfs:label` are declared inside it. For literal objects, such as Julian Hillyer and 2014-12-22T22:25:56.900Z, paired tags are used with

the objects being the content of the elements. Meanwhile, for resource object like affiliation, unpaired tag is used with the resource attribute is used to point to the relevant URI.

2.1.4.2 JSON-LD

The JSON-LD representation of the triples in Table 2.1 can be seen in Figure 2.4.

```

1  {
2    "@id": "http://orcid.org/0000-0002-3178-0201",
3    "@type": "http://xmlns.com/foaf/0.1/Person",
4    "http://purl.org/dc/terms/created": {
5      "@type": "http://www.w3.org/2001/XMLSchema#dateTime",
6      "@value": "2014-12-22T22:25:56.900Z"
7    },
8    "http://www.loc.gov/mads/rdf/v1#hasAffiliation": {
9      "@id": "http://www.grid.ac/institutes/grid.152326.1"
10   },
11   "http://www.w3.org/2000/01/rdf-schema#label": "Julian Hillyer"
12 }

```

Figure 2.4: JSON-LD for Triples in Table 2.1

In JSON-LD, the concept of grouping triples related to a certain entity is achieved using JSON objects. Specifically, the key in the JSON object refers to the predicate, and the value refers to the object. The value can be literal object or another JSON object. The use of JSON object as the value allows one to define literal object with specified datatype or non-literal object, i.e. the one that points to another URI (using @id). Meanwhile, the subject itself is defined within the same JSON object, with its ID specified as the value of @id and its type as the value of @type. Furthermore, the use of QName is also supported here. The namespace declaration is done in the JSON object as the value of the key @context. Another version of JSON-LD in Figure 2.4 with namespace declaration can be seen in Figure 2.5.

```

1  {
2    "@context": {
3      "foaf": "http://xmlns.com/foaf/0.1/",
4      "dcterms": "http://purl.org/dc/terms/",
5      "mads": "http://www.loc.gov/mads/rdf/v1#",
6      "rdfs": "http://www.w3.org/2000/01/rdf-schema#",
7      "xsd": "http://www.w3.org/2001/XMLSchema#",
8      "orcid": "http://orcid.org/"
9    },
10   "@id": "orcid:0000-0002-3178-0201",
11   "@type": "foaf:Person",
12   "dcterms:created": {
13     "@type": "xsd:dateTime",
14     "@value": "2014-12-22T22:25:56.900Z"
15   },
16   "mads:hasAffiliation": {
17     "@id": "http://www.grid.ac/institutes/grid.152326.1"
18   },
19   "rdfs:label": "Julian Hillyer"
20 }

```

Figure 2.5: JSON-LD with @context for Triples in Table 2.1

2.1.4.3 Turtle

The Turtle representation of the triples in Table 2.1 can be seen in Figure 2.6.

```

1  @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
2  @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
3  @prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
4  @prefix foaf: <http://xmlns.com/foaf/0.1/> .
5  @prefix dcterms: <http://purl.org/dc/terms/> .
6  @prefix orcid: <http://orcid.org/> .
7  @prefix mads: <http://www.loc.gov/mads/rdf/v1#> .
8
9  orcid:0000-0002-3178-0201 dcterms:created "2014-12-22T22:25:56.900Z"^^xsd:dateTime;
10   mads:hasAffiliation <http://www.grid.ac/institutes/grid.152326.1>;
11   rdf:type foaf:Person;
12   rdfs:label "Julian Hillyer".

```

Figure 2.6: Turtle for Triples in Table 2.1

Turtle representation usually starts with the namespace declaration to define namespace prefixes. Each namespace prefix declaration is indicated by the use of @prefix, which is followed by the prefix name and its URI. The triples are defined afterward by listing three resources in subject/predicate/object order with QNames. In Figure 2.6, the compact version of the syntax is used, with only the first triple is defined fully, the rest of triples only specify the predicates and the objects, since they share the same subject with the first one.

2.1.5 Taxonomy: RDF Schema (RDFS)

RDF Schema is considered as a semantic extension to RDF and commonly known as the schema language for RDF. It “extends” RDF by providing a set of distinguished resources

into the language (Allemang and Hendler, 2011) to describe properties and classes of RDF resources in a standardized manner. The main concepts of RDFS incorporating classes and properties. Resources can be divided into groups called classes, which the members of a class are known as the instances of a class, whereas properties describe relations between subject resources and object resources (W3C, 2014). Several examples of RDFS syntax can be seen in Table 2.3.

Syntax	Description
<code>rdfs:subPropertyOf</code>	Describes how two properties relate to each other
<code>rdfs:subClassOf</code>	Describes how two classes relate to each other
<code>rdfs:domain</code>	Refers to the subject of any triple that uses a certain predicate
<code>rdfs:range</code>	Refers to the object of any triple that uses a certain predicate
<code>rdfs:label</code>	Provides a printable name for any resource
<code>rdfs:seeAlso</code>	Specifies links of supplementary information about a resource
<code>rdfs:isDefinedBy</code>	Specifies a link to the primary source of information about a resource
<code>rdfs:comment</code>	Provides natural language comments about a model

Table 2.3: RDFS Syntax Examples (Allemang and Hendler, 2011)

2.1.6 Rule: Rule Interchange Format (RIF)

Rule Interchange Format (RIF) is a standard established by W3C for exchanging rules among different rule systems (Kifer and Boley, 2013). RIF was developed with the understanding that no single rule language could satisfy all the diverse needs of various paradigms in knowledge representation and business modelling. Consequently, RIF focuses on specifying rule exchange format rather than creating a universal rule language.

Taking this notion of rule exchange, the RIF Working Group designed a family of rule-based languages, called *dialects*, with specified syntax and semantics (Kifer and Boley, 2013). These dialects are designed to be both *uniform* and *extensible*. Uniformity ensures the dialects share as much common syntax and semantics as possible to facilitate an easier rule exchange, whereas extensibility allows experts to add new dialects as extensions of the existing ones, with the desired new functionalities.

The RIF Working Group has focused on two families of dialects: *logic dialects* and *production rule dialects* (Boley and Kifer, 2010). The logic dialects are designed to cover rule systems related to logic programming, deductive databases, and first-order logic, whereas the production rule dialects are designed to cover many commercial condition-action rule systems.

2.1.7 Ontology: Web Ontology Language (OWL)

In the Semantic Web framework, an ontology is a formal description of knowledge that outlines a domain's concepts and the relationships between them (Ontotext, nd). Following this, an ontology language is a language used to construct ontology.

Web Ontology Language (OWL) is the ontology language for the Web. It extends RDF and RDFS by providing additional vocabularies (terminologies) with formal semantics (DeWeese and Segal, 2014). Specifically, it extends the expresiveness of RDFS by allowing users to define classes, properties, instances (individuals), and meta-descriptions with more accuracy (Cur and Blin, 2015).

OWL provides three sublanguages: OWL Lite, OWL DL, and OWL Full. The latest version of OWL, the OWL 2, which published in 2009, includes many new features that extend the limited semantics of RDFS (Tuominen, 2020), for instance:

- equivalence (classes) and equality (individuals)
- disjointness (classes) and difference (individuals)
- boolean combination of classes
- local scope of properties
- special relational characteristics of properties
- cardinality of properties

2.1.8 Querying: SPARQL

Every data model has its own mechanism for retrieving data, and the RDF data model is no exception. To query and manipulate RDF data, SPARQL (SPARQL Protocol and RDF Query Language) is used. SPARQL's syntax is adopted from the well-known Structured Query Language (SQL) of relational database. Specifically, the `SELECT <result template> FROM <data set definition> WHERE <query pattern> query pattern`

of SQL is adapted for use in SPARQL (Cur and Blin, 2015). Nonetheless, the execution approach is different. SPARQL follows a navigational-based approach, where it starts at a specific node and traverses along certain edges to reach the target node. On the other hand, SQL approach relies on joins between tables (Cur and Blin, 2015).

The similarity between SQL and SPARQL lies within the semantic meaning of the syntax. In SPARQL, the `SELECT` clause defines which part of the query results will be retrieved. This is analogous to SQL, where columns to retrieve are listed in the `SELECT` clause. Both languages also use the `WHERE` clause to restrict the retrieved data to those that match the specified query patterns. However, the `FROM` clause—which is used to define the datasets to query—is optional in SPARQL. If there is no `WHERE` clause specified in the query, the dataset that will be used is the default graph. In addition, the optional `PREFIX` clause can also be used to declare abbreviations that will be used to reference URIs in the query (Cambridge Semantics, 2021). Lastly, query modifiers such as `ORDER BY`, `OFFSET`, `LIMIT`, `GROUP BY`, and `HAVING` can be applied to refine query results.

In defining query patterns in the `WHERE` clause, triples in the format of Turtle are used. These triples often contain variables, which store data from the RDF graph matching and can later be combined in other triples to form more complex queries. A variable is denoted by having a question mark (?), which followed by the variable's name. For instance, a variable with the name `user` is written as `?user`.

An example of SQL query with its corresponding SPARQL query is provided in Figure 2.7 to compare the syntaxes between the two. Both queries aim to retrieve the name and salary of the employees from the IT department in descending order.

	<code>PREFIX ex: <http://example.org/employees#></code>
<code>SELECT name, salary</code>	<code>SELECT ?name ?salary</code>
<code>FROM employees</code>	
<code>WHERE department = 'IT'</code>	<code>WHERE {</code>
	<code> ?employee ex:name ?name ;</code>
	<code> ex:salary ?salary ;</code>
	<code> ex:department "IT" .</code>
	<code>}</code>
<code>ORDER BY salary DESC;</code>	<code>ORDER BY DESC(?salary)</code>
SQL	SPARQL

Figure 2.7: SQL and SPARQL Syntax Comparison

The explanation for the SPARQL query in Figure 2.7 is as follows.

- The `ex` prefix is declared, which is associated with `http://example.org/employees#` to avoid the need to repeatedly write the full URI.
- The `SELECT` clause specifies the name of the variables to be returned. In this case, those variables are `?name` and `?salary`, which store the employee names and salaries respectively.
- The `WHERE` clause specifies the condition to be fulfilled. There are three match conditions here: it matches triples where the predicate is `ex:name` and binds the object to `?name`, it matches triples where the predicate is `ex:salary` and binds the object to `?salary`, as well as it matches triples where the predicate is `ex:department` and the object is the literal “IT”.
- The `ORDER BY` clause sorts the results by `?salary` in descending order.

2.1.9 Other Components

The previously-explained components—especially the RDF, OWL, and SPARQL—are the main building blocks of the Semantic Web. The next components to be discussed are the unifying logic, proof, trust, cryptography, and the user interface and applications, which add intelligence, verification, credibility, and usability to the Semantic Web.

The unifying logic and proof is a thinking framework set built on top of the ontology structure to enable them to derive new knowledge (Juneja and Sharma, 2017). Above that lies the trust layer, which addresses the credibility and reliability of the data and derived knowledge. The topmost layer with the highest layer is the user interface and applications layer. It serves as the medium for end-users to interact with the Semantic Web. Meanwhile, the layer that encapsulates all layers from the lowest layers, i.e., identifiers and character set layers until the proof layer is the cryptography layer. This layer ensures that the data is reliable, secure, and has not been altered during the transmission. Therefore, the integrity and authenticity of the information can be maintained perfectly. An example of this layer is the utilization of digital signatures for verification of the origin of the data sources (Juneja and Sharma, 2017).

2.2 Large Language Model (LLM)

Large Language Models (LLMs) are deep neural networks trained on vast amount of text data that are designed to understand, generate, and respond to human-like text.

This allows LLMs to capture deeper contextual information and subtleties of human language compared to traditional natural language processing (NLP) methods, such as rule-based systems, bag-of-words models, and many more. As a result, LLMs have significantly improved performance across various NLP tasks, such as text translation, sentiment analysis, question answering, and numerous others. Before the advent of LLMs, traditional methods excelled at categorization tasks and straightforward pattern recognition. Unfortunately, they typically underperformed in language tasks that required complex understanding and generation abilities. For example, previous generations of language models could not write an email from a list of keywords, which is a trivial task for contemporary LLMs (Raschka, 2024).

LLMs utilize an architecture called Transformer, which is a deep neural network architecture introduced by Vaswani et al. (2017) (covered in more detail in Subsubsection 2.2.1). This architecture enables them to pay selective attention to various parts of the input during predictions, making them particularly adept at managing the nuances and complexities of human language (Raschka, 2024).

2.2.1 General Architecture of Transformer

Before the inception of the Transformer, sequential architectures like Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997) used to be the state-of-the-art approaches in sequence modeling and transduction problems, such as language modeling and machine translation (Bahdanau et al., 2016; Cho et al., 2014; Sutskever et al., 2014). However, RNNs process data sequentially, which limits parallelization and slows down training, especially for long sequences. Furthermore, RNNs also struggle to learn dependencies between distant positions in sequences due to *vanishing gradient problem* (VGP), making it hard to capture long-term patterns (Hochreiter et al., 2001).

On the other hand, the Transformer is a model architecture that relies exclusively on attention mechanisms, eliminating the need for recurrence and convolutions. This mechanism is used to establish global dependencies between input and output. Therefore, this architecture enables much greater parallelization and eliminates the sequential problem in RNNs, making training process more efficient, especially for longer sequences. Unlike RNNs, Transformers capture long-range dependencies effectively

using self-attention mechanisms (covered in more detail in Subsection 2.2.1.3) (Vaswani et al., 2017).

The Transformer follows the architecture illustrated in Figure 2.8. It consists of two key components: the encoder (Subsubsection 2.2.1.1), shown on the left side of the figure and the decoder (Subsubsection 2.2.1.2), depicted on the right side. The encoder and decoder are composed of stacked self-attention and point-wise, fully connected layers. The encoder processes the input sequence, and the decoder generates the output sequence. At each step, the model operates auto-regressively (Graves, 2014), consuming the previously-generated symbols as additional input when generating the next.

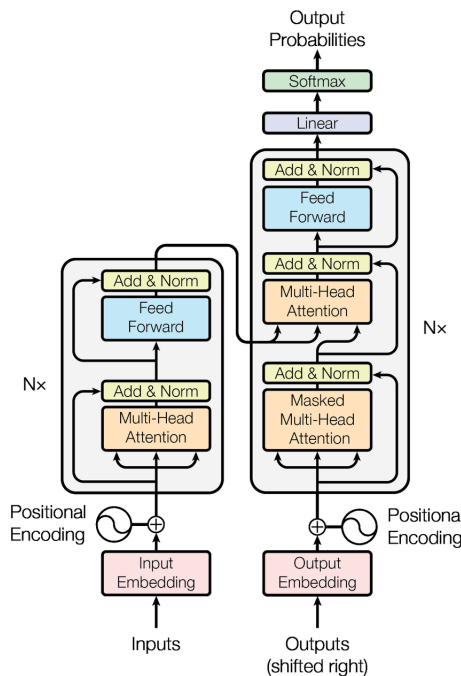


Figure 2.8: The Transformer's Architecture (Vaswani et al., 2017)

2.2.1.1 Encoder

The encoder is depicted on the left side of Figure 2.8. The encoder processes the input sequence and converts it into a continuous representation. It is composed of a stack of $N = 6$ identical layers, each with two sub-layers: a multi-head self-attention mechanism and a position-wise fully connected feed-forward network. Residual connections (He et al., 2015) and layer normalization (Ba et al., 2016) are applied around each sub-layer, i.e., the output of each sub-layer is computed as $LayerNorm(x + Sublayer(x))$, where $Sublayer(x)$ represents the function implemented by the sub-layer itself and x represents

the input of the sub-layer. Continuing, to support the residual connections, all sub-layers and embedding layers produce outputs with uniform dimension of $d_{model} = 512$ (Vaswani et al., 2017). These outputs are numerical representations of the input words, with each word assigned a unique dense vector based on its semantic meaning.

However, another challenge arises: a word can have multiple meanings depending on the context. Information about the relative or absolute position of the tokens in the sequence must be injected to solve this issue. Since the model contains no recurrence and no convolution, it needs a way to utilize the order of the sequence. To address this, Vaswani et al. (2017) added *positional encodings* to the input embeddings at the bottoms of the encoder and decoder stacks. These vectors provide context by indicating the position of each word within a sentence.

2.2.1.2 Decoder

The decoder is depicted on the right side of Figure 2.8. The decoder generates the output sequence from the encoder's representation. It is also composed of a stack of $N = 6$ identical layers, but with an additional sub-layer for multi-head attention over the encoder's output. Similar to the encoder, residual connections and layer normalization are also applied in the decoder. The decoder masks all the values for subsequent positions relative to a certain position to ensure the predictions for that position only depend on known outputs at previous positions, i.e., to preserve the auto-regressive property of the decoder (Vaswani et al., 2017).

2.2.1.3 Attention

The attention layer in the Transformer architecture is a mechanism that allows the model to focus on different parts of the input sequence when generating each part of the output sequence. An attention function maps a query and a set of key-value pairs to an output, where the query, keys, values, and output are all represented as vectors. The output is calculated as a weighted sum of the values, where the weight of each value is determined by a function which describes the compatibility of the query with the corresponding key (Vaswani et al., 2017). Therefore, it enables the model to dynamically weigh the importance of different input tokens with certain weighting factors.

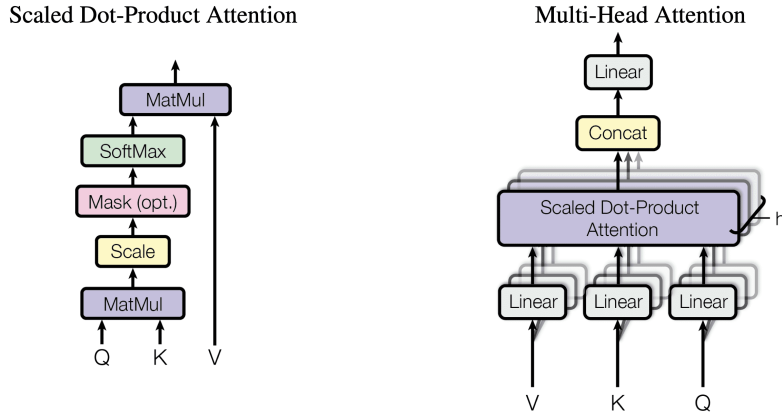


Figure 2.9: Scaled Dot-Product Attention and Multi-Head Attention (Vaswani et al., 2017)

The core component of an attention layer is called *scaled dot-product attention*, as depicted on the left side of Figure 2.9. Assume that the input consists of queries and keys with a dimension of d_k and values with the same dimension of d_v . It calculates the dot products between the query and all keys, divides each by $\sqrt{d_k}$ for standardization, and applies a softmax function to obtain the weights for the values. However, in practice, a set of n queries are computed simultaneously on the attention function, packed into a matrix $\mathbf{Q}$, where $\mathbf{Q} \in \mathbb{R}^{n \times d_k}$. The keys and values are also packed into matrices $\mathbf{K}$ and $\mathbf{V}$, where $\mathbf{K} \in \mathbb{R}^{m \times d_k}$, $\mathbf{V} \in \mathbb{R}^{m \times d_v}$, and m yields the number of key-value pairs (Vaswani et al., 2017). Hence, the matrix of outputs is computed as in Equation 2.1:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right) \mathbf{V} \in \mathbb{R}^{n \times d_v} \quad (2.1)$$

Instead of performing a single attention function with d_{model} -dimensional keys, values, and queries, Vaswani et al. (2017) proposed to use *multi-head attention*. With this mechanism, the queries, keys, and values are projected h times with different, learned linear projections to d_k , d_k , and d_v dimensions, respectively. It is possible by applying the attention function in parallel on each of their projected versions, yielding d_v -dimensional output values. These outputs are then concatenated and projected again to produce the final values, as depicted on the right side of Figure 2.9. Multi-head attention allows the model to join the information from different representation subspaces at different positions. It enhances the models ability to capture diverse features and therefore, improves performance.

2.2.2 Example Models

The advent of the Transformer architecture (Vaswani et al., 2017) marked a significant milestone, leading to the development of foundation models like BERT (Devlin et al., 2018), T5 (Raffel et al., 2019), and GPT (Radford et al., 2019). These models were built on this concept to adapt this architecture for different tasks (Raschka, 2024). We will explain those models along with other models utilized in this work subsequently.

2.2.2.1 BERT (Bidirectional Encoder Representations for Transformers)

BERT (Bidirectional Encoder Representations for Transformers) is a language representation model designed to pre-train deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context in all layers. BERT utilizes the Transformer encoder architecture (Subsubsection 2.2.1.1). It operates in two main steps: pre-training and fine-tuning (Devlin et al., 2018). During pre-training, the model is trained on unlabeled data, i.e., enormous amount of text, through two unsupervised tasks:

- **Masked Language Model (MLM):** Unlike traditional language models that predict the next word in a sequence, MLM randomly masks some tokens in the input by a special token (typically [MASK]) and predicts the original tokens based on the context provided by the surrounding words. This helps BERT learn the representations of individual words in context by taking preceding and subsequent words into account. For example, given the input “That [MASK] is flying in the sky”, the model must predict the word “plane” to ensure the sentence is coherent and meaningful.
- **Next Sentence Prediction (NSP):** NSP helps the model understand the relationship between two sentences, which is crucial for tasks like Question Answering (QA) and Natural Language Inference (NLI). During NSP training, the model is given pairs of sentences. In 50% of the cases, the second sentence follows the first one in the original text (labeled as `IsNext`), and in the other 50%, the second sentence is randomly selected from the corpus (labeled as `NotNext`). The model then has to correctly predict whether the second sentence follows the first one or not. For example, given the input a pair of two sentences, “The cat sat on the mat” and “It looked very comfortable”, the model has to predict that the latter sentence follows the former, since both sentences are related.

For fine-tuning, the BERT model is initialized with pre-trained parameters, which are then refined using labeled data specific to downstream tasks. Each downstream task has a separate fine-tuned model, even though they are initialized with the same pre-trained parameters. This approach allows BERT to achieve state-of-the-art results on various NLP tasks, such as question answering and language inference, without requiring substantial task-specific architectural modifications.

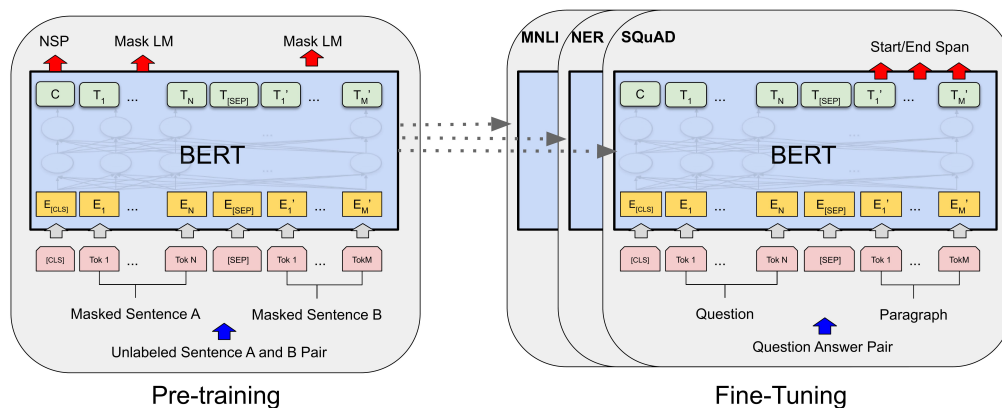


Figure 2.10: Pre-Training and Fine-Tuning in BERT (Devlin et al., 2018)

Figure 2.10 illustrates the pre-training and fine-tuning process with question-answering task as an example. Except for the output layers, the architectures used in pre-training and fine-tuning are identical. The same pre-trained model parameters are used to initialize models for different downstream tasks, and all parameters are fine-tuned during this process. The special symbol [CLS] is added at the beginning of every input example and [SEP] is used as a separator token (e.g., separating questions/answers).

2.2.2.2 T5 (Text-To-Text Transfer Transformer)

Another model based on the Transformer architecture is T5 (Text-To-Text Transfer Transformer). T5 introduces a unified framework where all NLP tasks are treated as text-to-text problems, meaning both inputs and outputs are handled as text sequences. For instance, to translate the sentence "That is good." from English to German, the model would be given the input "translate English to German: That is good." and trained to produce the output "Das ist gut." as illustrated in Figure 2.11. In text classification tasks, the model predicts a single word corresponding to the target label. This approach allows for the usage of the same model, objective, training procedure, and decoding process across a wide range of tasks (Raffel et al., 2019).

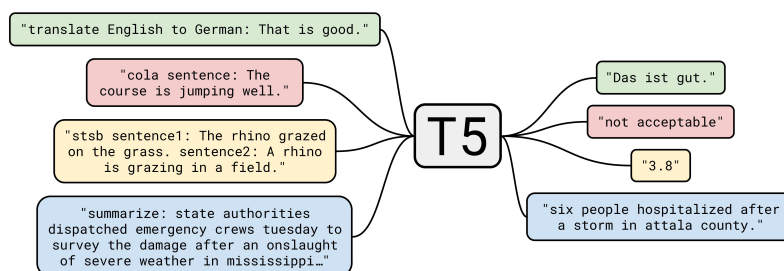


Figure 2.11: Usages of T5 (Raffel et al., 2019)

Unlike BERT (Subsection 2.2.2.1), which is an encoder-only model, T5 uses a standard encoder-decoder Transformer as proposed by Vaswani et al. (2017). The encoder converts the input text into a fixed-size vector representation, which the decoder then uses to generate the output text. While other modern approaches utilize a Transformer architecture with only a single “stack”, Raffel et al. (2019) found that employing a standard encoder-decoder structure achieved excellent results on both generative and classification tasks. The key innovation of T5 lies in its formulation of diverse tasks into a text-to-text format, enabling the same architecture to be applied across various tasks. T5 also provides a simple way to train a single model on a wide range of text tasks using the same loss function and decoding procedure. Despite its simplicity, T5 obtained comparable performance to task-specific architectures and ultimately produced state-of-the-art results when combined with scale.

2.2.2.3 GPT (Generative Pre-trained Transformer)

GPT (Generative Pre-trained Transformer) is a family of Transformer-based models introduced by OpenAI. This family has been crucial in advancing LLMs, powering generative AI applications like ChatGPT. NLP tasks such as question answering, machine translation, reading comprehension, and summarization are usually approached using supervised learning on specific datasets. However, Radford et al. (2019) demonstrated that language models could begin learning these tasks without explicit supervision when trained on a vast dataset of millions of webpages, known as WebText. This led to the development of the GPT-2 model, a Transformer with 1.5 billion parameters that achieved state-of-the-art results on 7 out of 8 language modeling benchmarks in a zero-shot (without example) setting. GPT-2 is a foundational model that paved the way for more advanced models like GPT-3 and GPT-4. GPT-2 demonstrated that increasing

the dataset size and the number of parameters significantly enhances a language models ability to comprehend tasks and outperform existing benchmarks in zero-shot scenarios. It also indicated that even larger language models could further improve natural language understanding (Kublik and Saboo, 2022).

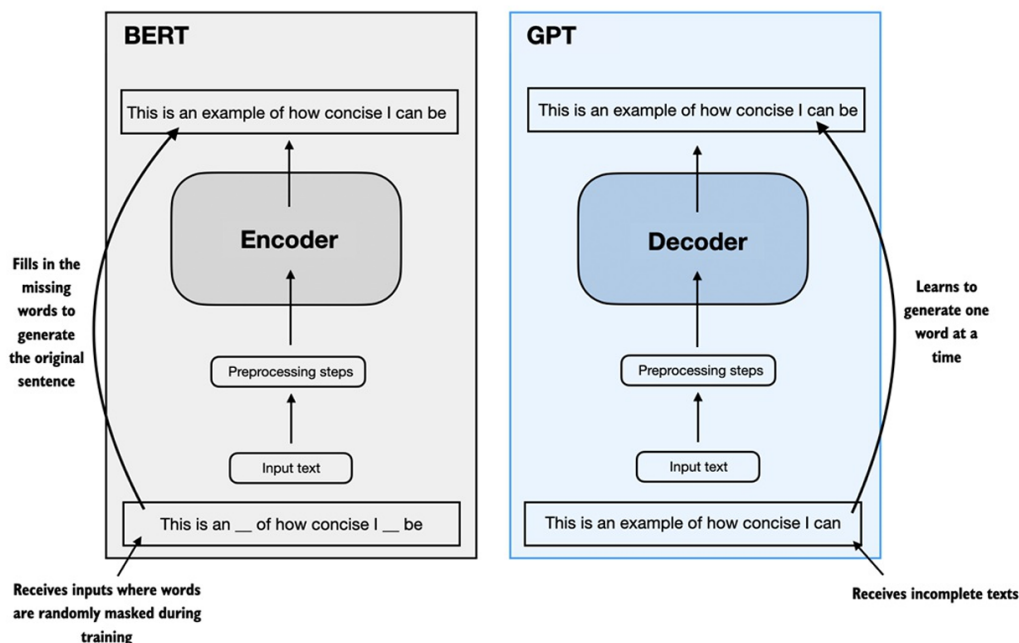


Figure 2.12: BERT vs. GPT (Raschka, 2024)

Unlike models such as BERT, which are designed for bidirectional understanding, GPT-2 is autoregressive, generating one token at a time based on the preceding tokens, i.e., next token prediction training scheme. This makes it particularly well-suited for tasks like text generation, completion, and summarization. As illustrated in Figure 2.12, GPT-2 is constructed using transformer decoder blocks (Subsubsection 2.2.1.2), in contrast to BERT, which utilizes transformer encoder blocks (Alammar, 2019).

To create a more robust and powerful language model, OpenAI developed GPT-3 (Brown et al., 2020), significantly larger than its predecessor, GPT-2, with 175 billion parameters and a dataset two orders of magnitude larger. Despite the increase in size, GPT-3 retains a similar architecture to GPT-2. It excels in various NLP tasks, even in zero-shot and few-shot (with few examples) settings. GPT-3 can generate human-like text and perform tasks it wasn't explicitly trained for, such as summing numbers, writing SQL queries, and coding in React and JavaScript based on plain English descriptions. While GPT-2 demonstrated some zero-shot abilities, GPT-3 can handle even more novel tasks when

provided with contextual examples (Kublik and Saboo, 2022).

The latest advancement of GPT model is GPT-4, which the first version of the technical report was published by OpenAI (2023). The model was released in March 14th, 2023. The same as its predecessors, GPT-4 leverages the Transformer model to generate text. However, the difference is that GPT-4 is *multimodal*, which can take both image and text as the inputs. This model outperforms most large language models and state-of-the-art systems after being evaluated on a suite of traditional NLP benchmarks (OpenAI, 2023).

2.2.2.4 MPNet

MPNet (Song et al., 2020) is a discriminative LLM that combines both MLM and permuted language modeling (PLM) in pre-training. It addresses both MLM's issue of ignoring masked tokens dependencies and PLM's issue of lacking position information in the permuted sentence. The main benefit of MPNet compared to BERT (Subsubsection 2.2.2.1) and XLNet (Yang et al., 2020) which rely solely on MLM and PLM, respectively, is its ability to condition on more contextual information when predicting masked tokens. This model has been pre-trained on large-scale text corpora as utilized in RoBERTa (Liu et al., 2019) pre-training and further fine-tuned on various downstream benchmarking tasks. Experiment results by Song et al. (2020) show that this model outperforms well-known MLM and PLM-based models on several benchmarks, such as GLUE (Wang et al., 2018), SQuAD (Rajpurkar et al., 2016), RACE (Lai et al., 2017), and IMDB (Maas et al., 2011).

The base model is published by Microsoft in HuggingFace² with MIT license. Several fine-tuned versions of the model by sentence-transformers using Sentence-BERT's approach (Reimers and Gurevych, 2019) are also available on HuggingFace, such as `all-mpnet-base-v2`³ for general purposes and `multi-qa-mpnet-base-cos-v1`⁴ for semantic search purposes.

²<https://huggingface.co/microsoft/mpnet-base>

³<https://huggingface.co/sentence-transformers/all-mpnet-base-v2>

⁴<https://huggingface.co/sentence-transformers/all-mpnet-base-v2>

2.2.2.5 LLaMA 3.1

LLaMA 3.1 (Dubey et al., 2024) is a set foundation models for language in three versions of parameters: 8B, 70B, and 405B, which support 128K context window length. These models are pre-trained on a corpus of 15T multilingual tokens with the use of decoder-only dense transformer architecture with next token prediction training, like in GPT (Subsubsection 2.2.2.3). Additionally, LLaMA 3.1 leverages grouped query attention with 8 key-value heads, an attention mask that prevents self-attention between different documents within the same sequence, a vocabulary consisting of 128K tokens, and the Rotary Positional Encoding (RoPE) (Su et al., 2024) which base hyperparameter has been increased to 500,000. The pre-training of this model consists of three stages: initial pre-training, long-context pre-training, and annealing. Several rounds of post-training are also applied, with each round composed of supervised fine-tuning (SFT) and Direct Preference Optimization (DPO) (Rafailov et al., 2024).

The open-source LLaMA 3.1 base models with several parameters options are available on HuggingFace: Llama-3.1-8B⁵, Llama-3.1-70B⁶, and Llama-3.1-405B⁷. Additionally, the corresponding instruction supervised fine-tuned version of the models are also available on HuggingFace: Llama-3.1-8B-Instruct⁸, Llama-3.1-70B-Instruct⁹, and Llama-3.1-405B-Instruct¹⁰. The most popular version of this model is the Llama-3.1-8B-Instruct which in November 2024 has been downloaded 6,192,025 times.

2.2.2.6 Jina Embeddings v3

Jina Embeddings v3 (Sturua et al., 2024) is a text embedding model developed by Jina AI. This model implements XLM-RoBERTa, a variant of BERT (as described in Subsubsection 2.2.2.1) that supports multilingual data with 570 million parameters. Several modifications are employed to enhance its ability to encode long text effectively, support task-specific encoding, and improve overall model efficiency. Specifically, this model leverages FlashAttention2 (Dao, 2023) for more efficient performance and RoPE

⁵<https://huggingface.co/meta-llama/Llama-3.1-8B>

⁶<https://huggingface.co/meta-llama/Llama-3.1-70B>

⁷<https://huggingface.co/meta-llama/Llama-3.1-405B>

⁸<https://huggingface.co/meta-llama/Llama-3.1-8B-Instruct>

⁹<https://huggingface.co/meta-llama/Llama-3.1-70B-Instruct>

¹⁰<https://huggingface.co/meta-llama/Llama-3.1-405B-Instruct>

(Su et al., 2024) to handle long text sequences. The output dimension of this model is flexibly determined by the user, with the default dimension of 1024, but it can be as low as 32 without sacrificing much performance thanks to the implementation of Matryoshka Representation Learning (MRL) (Kusupati et al., 2024).

The architecture for this model is illustrated in Figure 2.13. The model receives a document with a maximum of 8192 tokens and the task type, which can be `retrieval.passage`, `retrieval.query`, `separation`, `classification`, or `text-matching`. The core model is composed of a base `jina-XLM-RoBERTa` model and a task-specific LoRA (Hu et al., 2021) adapter to optimize the embeddings. To obtain the sentence embedding, mean pooling is used to aggregate the embeddings from each token. The training of this model involves three stages: pre-training the base model with MLM objective, fine-tuning the model for the embedding task with Sentence-BERT's contrastive approach (Reimers and Gurevych, 2019), and training the five distinctive task-specific LoRA adapters.

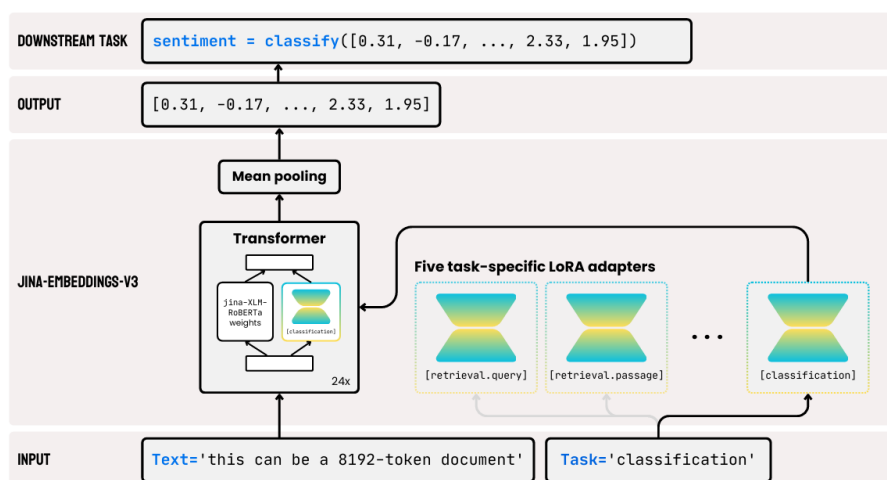


Figure 2.13: Jina Embeddings v3 Model Architecture (Sturua et al., 2024)

This model is available on HuggingFace, published by `jinaai` under the name of `jina-embeddings-v3` with the license of Creative Commons Attribution Non Commercial 4.0 (CC-BY-NC-4.0).

2.2.2.7 Mistral 7B

Mistral 7B (Jiang et al., 2023) is a decoder-only generative LLM with 7 billion parameters and 8192 context length developed by Mistral AI. In the implementation, it offers several improvements to the vanilla attention mechanism by incorporating grouped-query attention (GQA) (Ainslie et al., 2023) and sliding window attention (SWA) (Beltagy et al., 2020; Child et al., 2019). GQA accelerates the inference speed and decreases the amount of memory needed for decoding, whereas SWA enables long sequence handling. The model is further fine-tuned on instruction datasets to create the Mistral 7B – Instruct model. This fine-tuned model outperforms all 7B models on MT-Bench and is comparable to 13B chat (instruction fine-tuned) models.

The base models of Mistral 7B are available on HuggingFace in two versions: Mistral-7B-v0.1¹¹ and Mistral-7B-v0.3¹². Notably, Mistral-7B-v0.2 was not released, as one of the team members mentioned they inadvertently forgot to do so¹³. For the instruction fine-tuned model, the three versions are available on HuggingFace: Mistral-7B-v0.1-Instruct¹⁴, Mistral-7B-v0.2-Instruct¹⁵, and Mistral-7B-v0.3-Instruct¹⁶. All of these models are released under the Apache 2.0 License, which indicates that the models can be used without restrictions.

2.2.2.8 Mistral NeMo

Mistral NeMo (Mistral AI Team, 2024) is a 12B parameter model developed through a collaboration between NVIDIA and Mistral AI, featuring support for a 128K context window. It is tailored for multilingual tasks, demonstrating strong performance in languages such as English, French, German, Spanish, Italian, Portuguese, Chinese, Japanese, Korean, Arabic, and Hindi. The model employs the Tekken tokenizer, which is based on Tiktoken, and trained on over 100 languages. This tokenizer has shown improved efficiency, achieving a better compression for approximately 85% of all languages.

¹¹<https://huggingface.co/mistralai/Mistral-7B-v0.1>

¹²<https://huggingface.co/mistralai/Mistral-7B-v0.3>

¹³<https://x.com/dchaplot/status/1772489690341605472>

¹⁴<https://huggingface.co/mistralai/Mistral-7B-v0.1-Instruct>

¹⁵<https://huggingface.co/mistralai/Mistral-7B-v0.2-Instruct>

¹⁶<https://huggingface.co/mistralai/Mistral-7B-v0.3-Instruct>

There are two versions of this model: the base model (Mistral-Nemo-Base-2407¹⁷) and the instruction fine-tuned model (Mistral-Nemo-Instruct-2407¹⁸). Both are hosted on HuggingFace and released under the Apache 2.0 License, just like the Mistral 7B (Subsubsection 2.2.2.7).

2.2.2.9 Qwen2.5

Qwen2.5 is a series of dense, decoder-only, open-source LLMs created by Alibaba Cloud. During its pre-training, the model utilizes a vast and up-to-date dataset consisting of up to 18 trillion tokens. It also supports a context window of up to 128K tokens and can produce up to 8K tokens. In comparison to its predecessor, Qwen2, Qwen2.5 offers enhanced capabilities in coding and mathematics tasks, along with a broader knowledge base (Qwen Team, 2024). Additionally, there are two domain expert LLMs based on this model: Qwen2.5-Coder (explained in Subsubsection 2.2.2.10) for coding and Qwen2.5-Math for mathematics.

There are 7 different sizes of Qwen2.5, all hosted on HuggingFace: 0.5B¹⁹, 1.5B²⁰, 3B²¹, 7B²², 14B²³, and 32B²⁴. Each size of the model has its own instruction fine-tuned version, which is denoted by the `-Instruct` suffix in the model name. These models are licensed under Apache 2.0, except for the 3B version which is under Qwen Research license and the 72B which is under Qwen license.

2.2.2.10 Qwen2.5-Coder

Qwen2.5-Coder (Hui et al., 2024) is a family of domain expert (coding tasks) LLMs released by Alibaba Cloud, based on Qwen2.5's architecture and tokenizer. These models are trained on extensive datasets of 5.5T tokens and fine-tuned on instruction datasets curated for coding tasks. Specifically, the training pipeline consists of three stages, as visualized in Figure 2.14. In the first stage, the model learns from individual code files through the next token prediction and fill-in-the-middle (Bavarian et al., 2022), with a

¹⁷<https://huggingface.co/mistralai/Mistral-Nemo-Base-2407>

¹⁸<https://huggingface.co/mistralai/Mistral-Nemo-Instruct-2407>

¹⁹<https://huggingface.co/Qwen/Qwen2.5-0.5B>

²⁰<https://huggingface.co/Qwen/Qwen2.5-1.5B>

²¹<https://huggingface.co/Qwen/Qwen2.5-3B>

²²<https://huggingface.co/Qwen/Qwen2.5-7B>

²³<https://huggingface.co/Qwen/Qwen2.5-14B>

²⁴<https://huggingface.co/Qwen/Qwen2.5-32B>

maximum context length of 8,192 tokens. The next stage aims to enhance the model's long-context capabilities up to 131,072 tokens. Lastly, supervised fine-tuning with the instruction dataset and direct preference optimization (DPO) (Rafailov et al., 2024) are conducted for further alignment of the LLM.

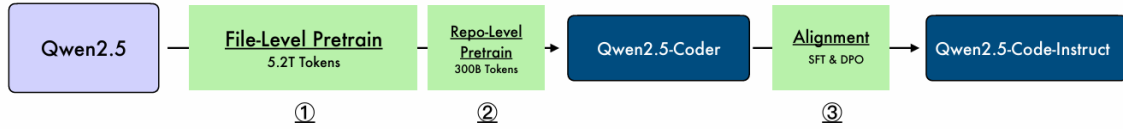


Figure 2.14: Qwen2.5-Coder Training Pipeline (Hui et al., 2024)

Just like Qwen2.5, Qwen2.5-Coder is publicly accessible through HuggingFace and it comes in 7 different sizes: 0.5B²⁵, 1.5B²⁶, 3B²⁷, 7B²⁸, 14B²⁹, and 32B³⁰. Likewise, each base model has its instruction fine-tuned version, which is denoted by the `-Instruct` suffix in the model name. The same license, Apache 2.0, applies for all models, except for the 3B which is licensed under the Qwen Research license.

2.3 Retrieval-Augmented Generation (RAG)

Pre-trained large language models have successfully shown their capabilities in learning vast amount of knowledge. These models store the knowledge they have learned in their parameters, without utilizing any other external memory; known as parameterized implicit knowledge base (Raffel et al., 2019; Roberts et al., 2020, as cited in Lewis et al., 2021). Whilst they do fine on downstream NLP tasks (Choi et al., 2021), these models have several major limitations, as stated by Lewis et al. (2021). Firstly, it is very hard to expand or revise their knowledge. One might assume that fine-tuning an LLM on domain-specific knowledge would enable it to adapt to the new data effectively while retaining the information acquired during pre-training. However, this is not always the case. Fine-tuning LLMs on domain-specific knowledge may lead to performance degradation, known as *catastrophic forgetting* (Kaushik et al., 2021). In this phenomenon, a model's recent learning overshadows its previously acquired knowledge, causing a major decline in performance on previous tasks (Liu et al., 2024).

²⁵<https://huggingface.co/Qwen/Qwen2.5-Coder-0.5B>

²⁶<https://huggingface.co/Qwen/Qwen2.5-Coder-1.5B>

²⁷<https://huggingface.co/Qwen/Qwen2.5-Coder-3B>

²⁸<https://huggingface.co/Qwen/Qwen2.5-Coder-7B>

²⁹<https://huggingface.co/Qwen/Qwen2.5-Coder-14B>

³⁰<https://huggingface.co/Qwen/Qwen2.5-Coder-32B>

The second problem is also encountered by most state-of-the-art deep learning models nowadays. This problem is commonly referred as *the black-box problem* of AI (von Eschenbach, 2021), due to the lack of transparency and interpretability in deep learning models. The inference processes in these models, i.e., how these models reach conclusions, are opaque or hidden. Without a proper understanding of these matters, it is still not clear to what extent one can trust those AI systems (von Eschenbach, 2021). Consequently, such systems sometimes are insufficiently reliable for decision-making, particularly in critical areas, such as security, healthcare, and safety.

The last problem is about the “hallucinations” LLMs can exhibit. Hallucination refers to a condition where an LLM generates factually incorrect information, yet sounds plausible (Wei et al., 2024b). This phenomenon occurs if LLMs encounter questions which answers incorporate knowledge that has not been trained on them before (Hao et al., 2024). Even though this phenomenon may seem inevitable due to the evolving nature of knowledge, there are some strategies that can be used to reduce hallucinations in LLMs. This includes the use of *retrieval-augmented generation* (RAG) approach to generate more reliable responses.

Instead of relying solely on parametric memory, RAG takes a hybrid approach by combining parametric and non-parametric (i.e., retrieval-based) memory. The use of non-parametric memory is vital here, as it provides a knowledge base that can be expanded and revised, and the accessed knowledge can be inspected and interpreted (Lewis et al., 2021). In short, RAG is a technique that retrieves information from the non-parametric memory based on the user’s query and utilizes the retrieved results to generate AI responses using parametric memory (Larson and Truitt, 2024). This concept is implemented in frameworks such as LlamaIndex³¹ and LangChain³².

2.3.1 RAG vs. Fine-tuning

The enhancement of LLMs has attracted significant interest due to their increasing prevalence. Among the optimization techniques, RAG is frequently compared with *fine-tuning* (FT). RAG excels in dynamic environments by offering real-time knowledge updates and effectively leveraging external sources, though it comes with higher latency

³¹<https://www.llamaindex.ai/>

³²<https://www.langchain.com/>

and ethical considerations. On the other hand, FT enables deep customization of a models behavior and style, but requires substantial computational resources and re-training for updates. Apart from the *catastrophic forgetting* phenomenon discussed in Subsection 2.3, various evaluations of performance on diverse knowledge-intensive tasks have shown that RAG consistently outperforms unsupervised fine-tuning. Ovadia et al. (2024) demonstrated that although unsupervised fine-tuning offers some improvements, RAG excels in handling both previously-encountered knowledge and entirely new information. Furthermore, it was observed that LLMs struggle to learn new factual information through unsupervised fine-tuning. Therefore, the choice between RAG and FT depends on the specific requirements for data dynamics, customization, and computational resources in the application context. Both methods can complement each other, enhancing a models capabilities at different levels (Gao et al., 2024).

2.3.2 Components

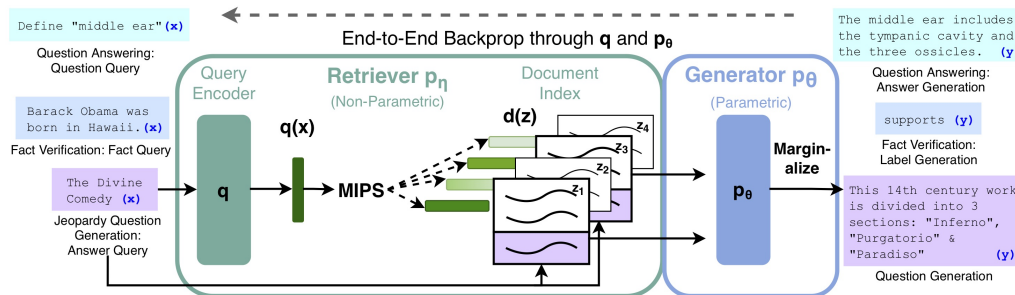


Figure 2.15: RAG Approach Overview (Lewis et al., 2021)

RAG has two components corresponding to the types of memory used: *retriever* and *generator*. These components work together as shown in Figure 2.15. A pre-trained retriever, composed of a *query encoder* and a *document index*, is combined with a pre-trained sequence-to-sequence (seq2seq) model as the generator. The flow of RAG as proposed by Lewis et al. (2021) is straightforward: given a query x , it is first encoded into dense representation $q(x)$ using the *query encoder* q , which is based on the BERT_{BASE}. Maximum Inner Product Search (MIPS) is then employed to retrieve a set of top- K documents $\{z_i: z_i \in Z; 1 \leq i \leq K\}$ from a collection of documents (corpus) Z utilizing the dense representation of z , i.e. $d(z)$ which is also produced by a BERT_{BASE} *document encoder* d . For the final prediction y , z is treated as a latent variable, and the seq2seq models predictions are marginalized over the different retrieved documents (Lewis et al.,

2021). Each component will be explained subsequently.

2.3.2.1 Retriever

The retriever in RAG corresponds to the non-parametric memory. The choice of retriever typically depends on the type of retrieval source, which can include text, semi-structured data (e.g., PDF files), and structured data (e.g., knowledge graphs). In the architecture proposed by Lewis et al. (2021)—which is usually referred to as naïve or vector-based RAG—a Dense Passage Retriever (DPR) was used, although different retrievers may be employed based on the type of the retrieval source. Generally, for text retrieval, the retriever is classified into three categories.

1. **Sparse retriever (keyword search):** This type of retriever relies on exact keyword matching between the terms in the query and documents. A more advanced version like BM25 (Robertson and Zaragoza, 2009) incorporates factors like term frequency (TF), inverse of document frequency (IDF), and specific weighting schemes to enhance retrieval effectiveness. In this approach, both query and documents are encoded into the same BM25 vector space, and cosine similarity is used to measure the relevance between them. A higher similarity score indicates a greater relevance of the document to the query.
2. **Dense retriever (dense vector search):** This retriever behaves similarly to the sparse retriever, except that it uses lower-dimensional dense representations instead of sparse ones. Discriminative LLMs, such as BERT (Devlin et al., 2018) or MPNet (Song et al., 2020) are often used to generate these dense embeddings. Likewise, relevance is then determined by computing cosine similarity between the query and document embeddings. DPR (Karpukhin et al., 2020) is based on this approach, where two BERT models (each for query and document) are fine-tuned such that the inner product (which is proportional to cosine similarity) values for relevant query-document pairs are high, and vice versa. A simpler approach is to use off-the-shelf discriminative LLMs which have already been fine-tuned on query-document pairs to encode the query and documents.
3. **Hybrid retriever (hybrid search):** This retriever combines both keyword search and dense vector search. For instance, an approach by Weaviate³³, the relative score fusion

³³<https://weaviate.io/blog/hybrid-search-fusion-algorithms>

algorithm scales the similarity scores from both sparse and dense retrievers to $[0, 1]$. The final relevance score is then calculated as the scaled sum of these two scores, effectively leveraging the strengths of both retrieval approaches.

2.3.2.2 Generator

The posed query and selected documents from the retriever are combined into a coherent prompt, to which a generator, i.e., LLM is given a task to formulate the response. The generator component can be modeled using any encoder-decoder architecture (Lewis et al., 2021). The models approach to answering may vary, depending on the task-specific criteria, allowing it to either draw upon its inherent parametric knowledge or restrict its responses to the information contained within the provided documents. For ongoing dialogues, any existing conversational history can be integrated into the prompt, enabling the model to effectively engage in multi-turn dialogue interactions (Gao et al., 2024).

In the architecture proposed by Lewis et al. (2021), BART-large (Lewis et al., 2019), a pre-trained seq2seq transformer with 400 million parameters, serves as the generator. When generating response from BART, the query input x is concatenated with the retrieved content z .

2.3.3 GraphRAG

GraphRAG is a RAG system that leverages KG as the knowledge base. The implementation involves retrieving graph elements that contain knowledge pertinent to the given query from a graph database (Peng et al., 2024). GraphRAG provides several advantages over naïve RAG, including the ability to capture connections between entities for a more comprehensive retrieval of relational information, to represent knowledge in a more efficient and effective format rather than lengthy passages, and to capture a broader context of information, which enables answering *query-focused summarization* (QFS) tasks (Peng et al., 2024).

The primary distinction between naïve RAG and GraphRAG is their knowledge bases and retrievers. Generally, naïve RAG relies on textual data as the knowledge base and utilizes DPR as the retriever. In contrast, GraphRAG can employ various types of retrievers to perform retrieval from a KG as its knowledge base. Specifically, Peng et al. (2024) classify graph retrievers into three main categories, as follows.

1. **Non-parametric retriever**, which does not incorporate deep learning models to perform retrieval. Instead, it leverages heuristic rules and graph search algorithms. For instance, multihop paths retrieval over the topic entities can be employed to capture answer entities (Taunk et al., 2023; Yasunaga et al., 2021).
2. **LM-based retriever**, which incorporates LM in the retrieval due to its outstanding natural language understanding capabilities. For instance, RoBERTa (Liu et al., 2019) can be trained as a retriever by expanding the paths from the topic entities and retrieving relevant paths through a sequential decision process (Zhang et al., 2022). Another approach, KG-GPT (Kim et al., 2023), integrates an LLM to produce top- K relevant relations given a specific entity.
3. **Graph neural network (GNN)-based retriever**, which incorporates GNN to comprehend the structure of the graph effectively. For instance, the approach by Mavromatis and Karypis (2024), GNN-RAG, leverages the encoded graph to retrieve relevant entities based on the given query.

Apart from the aforementioned approaches, another approach is to perform retrieval by querying the graph, i.e., query-based GraphRAG, where the query language can be either SPARQL for RDF data model or Cypher for Neo4j. In this approach, query generation is the most critical task and can be accomplished through manual and semi-automatic approaches, schema-based approaches, or other approaches that incorporate deep learning (DL) (Rony et al., 2022). However, recent developments in DL and LLM have shown a growing interest in the latter approach. For instance, Rony et al. (2022) introduced SGPT that leverages a stack of Transformer encoders to encode both linguistic features of a question and its subgraph information which are then provided into GPT-2 (Radford et al., 2019) to generate the appropriate query. Few other approaches (Emonet et al., 2024; Kovriguina et al., 2023) do not require training at all, leveraging in-context learning only with n -shot prompting.

This query-generation approach provides enhanced flexibility to address a wide range of questions, especially questions that require information based on the structure of the graph or a broader context of the graph rather than just a conventional text retrieval. Consider the question posed in Figure 2.16, “How many children does Lionel Messi have?” By leveraging RDF-based KG, the answer to the question can be easily obtained using SPARQL query execution. In contrast, text retrieval struggles in such cases. The retriever

will perform accurately iff there exists a document chunk that explicitly states Lionel Messi has three children. Otherwise, consider the situation illustrated in Figure 2.16, where information about Lionel Messi’s children is spread into three different chunks. If the retriever only selects the top-2 chunks, then the generator may conclude that Lionel Messi has only two children, omitting the third chunk which contains information about his other child. Additionally, by not providing the “explicit” knowledge to the generator and letting the generator infer the number of children that Lionel Messi has, the likelihood of the generator hallucinating will be subsequently increased. Thus, this approach heavily relies on the top- k retrieved chunks and their contents, which might not always explicitly reflect the complete information needed.

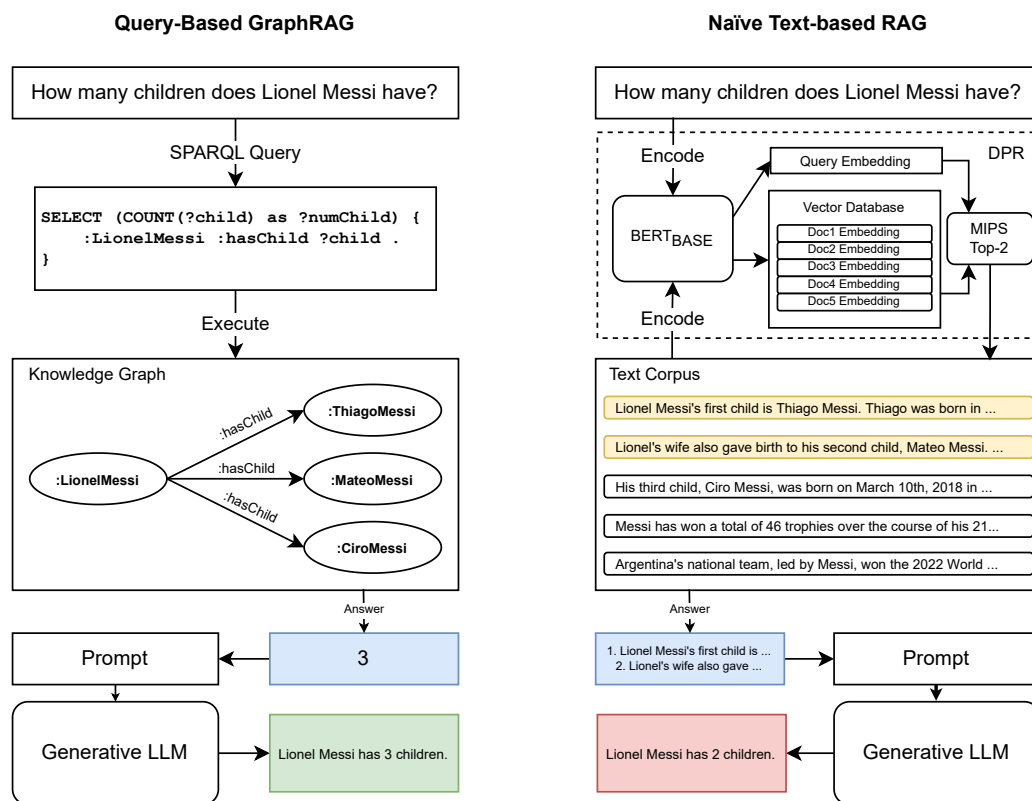


Figure 2.16: Query-based GraphRAG vs. Naïve Text-based RAG

2.4 Comparison of Several QA Systems based on KG or LLM

There are several types of QA systems based on KG or LLM: KG-based QA systems, LLM-based QA systems, text-based RAG QA systems, and GraphRAG QA systems.

2.4.1 KG-based QA Systems

Traditional QA systems that are based on KG and do not incorporate LLMs mainly leverage pre-defined templates or rules to parse questions into logical forms (Fu et al., 2020). Several approaches include using a bottom-up parser (Berant et al., 2013), mapping questions to templates which are then ranked to obtain the answers (Bast and Haussmann, 2015), and leveraging an automated template generation model which utilizes dependency parsing and rule-based methods to generate query and question templates to be ranked to provide the final answer (Abujabal et al., 2017). Additionally, other approaches choose to translate the natural language question into a SPARQL query, which can then be used to retrieve the desired information (Diefenbach et al., 2018). The latter technique is mainly composed of five tasks: question analysis, phrase mapping, disambiguation, query construction, and querying distributed knowledge.

Intuitively, this approach costs the lowest compared to other approaches that leverage LLMs in the process, due to the computational efficiency of pre-defined rules and template-based methods, which typically do not require the large-scale computational power typically associated with LLMs. Since LLMs perform autoregressively, the computational time is also often slower than traditional approaches. However, the trade-off is a lack of flexibility and adaptability. Since the system heavily depends on pre-defined templates and rules, it can struggle with handling more complex questions that do not fit into the predefined structures. This rigidity limits the system's ability to generalize across a wide variety of question types, especially those involving diverse queries, thus reducing its effectiveness in dynamic scenarios where questions are often more varied and nuanced.

2.4.2 LLM-based QA Systems

The emergence of LLMs have transformed the QA landscape by enabling systems to understand and generate natural language text with a high degree of accuracy and fluency. Unlike traditional KG-based QA systems that rely on pre-defined templates or rules, LLM-based QA systems fully rely on the vast knowledge encoded in the parameters of the models as a result of being trained on large, diverse datasets. This allows LLM-based systems to handle a broader range of questions, including those that are more complex and open-ended.

However, LLM-based QA systems face several problems related to the black-box nature of LLMs, difficulty in expanding or revising the knowledge, and may produce hallucinations, where the model outputs a plausible but nonfactual responses (Lewis et al., 2021). Thus, despite they perform well in general natural language tasks, they might be not reliable in knowledge-intensive tasks.

As explained before, the computational cost of training and deploying LLMs is significant, requiring substantial resource consumption. In terms of latency, LLMs tend to generate responses more slowly than traditional methods due to the autoregressive nature, where each token is generated sequentially.

2.4.3 Text-based RAG QA Systems

RAG (Lewis et al., 2021) was introduced to remedy the limitations of LLMs for QA on knowledge-intensive tasks, by combining retrieval model with LLMs. This approach improves the accuracy and reliability of answers by allowing the system to retrieve relevant information as the context before generating responses. Through this method, the more up-to-date knowledge can be incorporated in the generation as it is easier to update the knowledge stored in the retrieval model compared to the LLM.

Despite this, text-based RAG systems have several drawbacks. One major challenge is the interpretability of the text representations used in the retrieval process. Vector representations (i.e., embeddings) of text are encoded as arrays of real numbers, making them inherently opaque and difficult to interpret. This lack of transparency complicates the understanding of how the retrieval model identifies and selects relevant documents.

Additionally, the retrieval process requires auxiliary storage to accommodate the corpus of documents and substantial computational resources to encode, index, search, and rank large datasets effectively. These demands are further exacerbated by the use of discriminative neural language models for encoding documents, like in DPR (Karpukhin et al., 2020). While these models consume fewer resources compared to massive generative models, they still require more computational power than traditional ranking algorithms. Moreover, advanced techniques such as learning-to-rank models are often employed to rerank retrieved documents, which further increases the computational time and the required memory.

Lastly, while the RAG can reduce hallucinations compared to pure LLM-based systems, it does not completely eliminate them, as the LLM still plays a significant role in generating the final output. The final response may still include fabricated information, even when the retrieved information is valid. This limitation arises because RAG systems still rely on the generator (i.e., LLM) to comprehend the retrieved documents and extract relevant information to answer the specific question posed by the user. The generation process can introduce errors, particularly if the LLM misinterprets the retrieved content.

2.4.4 GraphRAG QA Systems

GraphRAG QA systems, particularly those that leverage query-based retrieval mechanisms, address several limitations observed in text-based RAG systems. Firstly, since these systems generate structured queries (e.g., SPARQL) to interact with the KG and retrieve the relevant answers. This approach significantly enhances the interpretability of the retrieval process, as the queries explicitly describe the relationships and entities involved in obtaining the answer. Unlike vague vector embeddings, the query structure is more transparent and human-readable, allowing for easier debugging and validation of the system's reasoning process.

Secondly, querying a KG using SPARQL is faster and more resource-efficient compared to the document ranking process in text-based RAG systems. KG queries directly retrieve relevant information through a navigational-based approach (i.e., graph traversal) (Cur and Blin, 2015). In contrast, text-based RAG systems require embeddings generation, similarity computations, and advanced reranking, all of which are computationally intensive and time-consuming. Moreover, the compact nature of RDF-based KGs—which are stored as triples—reduces memory requirements compared to the extensive storage needed for document corpora and embeddings in text-based systems.

Lastly, these systems can “bypass” the need for LLMs to interpret retrieved text documents. The execution of SPARQL queries directly retrieves the specific information required to answer a question. This contrasts with text-based RAG systems, where the LLM must process and comprehend unstructured text documents to extract relevant information, which can introduce potential errors due to hallucinations.

Despite all these advantages, the problem regarding interpretability still remains within

the GraphRAG systems, albeit in different forms. While structured queries like SPARQL provide transparency, the translation of a natural language question into a SPARQL query often relies on complex query generation mechanisms. While traditional methods like rule-based methods or template-based methods may not encounter this problem, they lack flexibility and adaptability for handling diverse questions. On the other hand, employing LLMs for this translation task significantly enhances flexibility and adaptability but reintroduces interpretability issues due to their black-box nature.

While LLMs can offer more flexible and context-aware query generation, their usage significantly increases computational overhead. This is because generating SPARQL queries typically requires multiple stages, such as question analysis, entity recognition, relation linking, and query construction, all of which may involve LLM inference. The autoregressive nature of LLMs further adds to processing time, especially when compared to simpler deterministic methods. This complexity also increases the risk of error propagation, where mistakes in earlier stages of query generation can cascade, affecting the accuracy of the final result. Therefore, while GraphRAG systems excel in efficiency and transparency in certain aspects, the reliance on intricate query generation mechanisms can pose significant challenges in terms of resource usage and system reliability.

In conclusion, despite the two aforementioned limitations, we chose a GraphRAG QA system primarily for its transparency and reduced risk of hallucination, which are critical for ensuring reliability in high-stakes tasks. Compared to text-based RAG systems which rely on less-interpretable embeddings, GraphRAG leverages structured queries like SPARQL, providing a more clear and interpretable process for retrieving information. This transparency is very important for debugging, validation, and trust in the system's reasoning. Furthermore, GraphRAG minimizes hallucinations by generating answers directly from precise data retrieved from the KG, eliminating the need for LLMs to interpret long, unstructured text—a task where LLM's performance tends to degrade (Hosseini et al., 2024). These qualities make GraphRAG an optimal choice for applications demanding high reliability and accountability.

2.4.5 Performance and Accuracy Comparison

While there is no work that directly accounts for the comparison between all the previously mentioned systems, Vrandečić (2023) compared the use of an LLM

(ChatGPT), a search engine (Google), and a KG (Wikidata) for answering the question “Who created The School of Athens?” As discussed in Subsubsection 2.4.2, ChatGPT, despite running on the fastest hardware available, took approximately 5 seconds to answer the question, with almost one second spent parsing and processing the question before generating a response. However, ChatGPT did not provide a direct answer; instead, it generated extensive contextual information that was not requested. On the other hand, Google took roughly 0.5 seconds to retrieve 27,900,000 results and generate a summary from the relevant result. Lastly, a relevant SPARQL query was executed over the Wikidata Query Service, retrieving the corresponding result in nearly 0.2 seconds. We henceforth will use this information in the analysis of the computational time.

Rooting from the previous analysis on the comparison results, it can be inferred that conventional KG-based QA systems will generally offer the lowest latency due to the usages of templates or rules without any advanced machine learning algorithms. However, there is no existing work that directly addresses the computational time for such systems in a comprehensive manner. Therefore, based on the execution time of a SPARQL query, we can reasonably estimate that the latency of such systems will be greater than 0.2 seconds. The exact time will depend on factors such as the complexity of the query, the underlying KG, and the system implementation.

For LLM-based QA systems, the latency fully depends on the LLM itself. Based on previous explanation, we can estimate the latency to be around 5 seconds, though this can vary in practice depending on factors such as the number of generated tokens and the hardware specifications. On the other hand, the latency of text-based RAG is a combination of both the retrieval and generation processes. The retrieval phase typically takes around 0.5 seconds, while the generation phase, which involves the LLM processing the retrieved context and generating the answer, takes approximately 5 seconds. Therefore, the total latency for text-based RAG systems is approximately 5.5 seconds. However, because text-based RAG systems often provide a longer document context to the LLM, the generation phase may take slightly longer time. Consequently, the overall latency for text-based RAG systems is likely to exceed 5.5 seconds. Lastly, for GraphRAG systems, which typically involve multiple LLM calls, the latency can be much higher. At least two LLM calls are involved in this pipeline: one for SPARQL query generation and another for answer generation. Combined with the query execution,

the overall ' latency will likely exceed 10.2 seconds, though it can be shorter for simpler questions that require a shorter SPARQL query and a more concise answer.

In terms of memory consumption, systems which leverage LLMs tend to have significantly higher memory requirements. Particularly, GPUs with massive VRAM capacity are needed. For instance, to load a 8B model in the full form, i.e., in FP16, 16GB VRAM is typically required (Schmid et al., 2024). To represent knowledge, RDF triples are more concise compared to raw text. In GraphDB³⁴, 9GB of RAM is needed to load 150M of triples³⁵. This level of storage efficiency is far superior when compared to storing knowledge in unstructured text, where the verbosity of natural language and the lack of direct associations between concepts can cause significant memory overhead. As a result, raw text storage becomes more resource-intensive, especially for large-scale knowledge bases, highlighting the memory efficiency of RDF for structured data representation. With the assumption of using UTF-8 encoding, a sentence composed of 20 words³⁶, and a word composed of 5 characters³⁷, 9GB of memory can only store approximately 96M sentences. The calculation steps are as follows.

- Using UTF-8 encoding, a single character typically requires 1 byte of memory.
- Assuming that each word consists of an average of 5 characters, a single word requires 5 bytes.
- Assuming that a sentence is composed of 20 words, the size of a single sentence is calculated as: $5 \times 20 = 100$ bytes.
- Since 1 GB = 1073741824 bytes, then 9 GB = 9663676416 bytes.
- Hence, the total number of sentences that can fit into 9GB of memory is $\left\lfloor \frac{9663676416}{100} \right\rfloor = 96636764 \approx 96\text{M}$.

Thus, if we were to rank the memory requirements from the most demanding to the least, the order would be: GraphRAG, text-based RAG, LLM-based QA, and KG-based QA.

Lastly, in terms of accuracy, RAG systems outperform those relying solely on parametric models (i.e., closed-book QA) for answering questions on various benchmarking datasets (Lewis et al., 2021). Moreover, experiments conducted by Gosnell and de Saint Pern

³⁴<https://graphdb.ontotext.com/>

³⁵<https://graphdb.ontotext.com/documentation/10.8/requirements.html>

³⁶<https://languagetool.org/insights/post/sentence-length/>

³⁷<https://charactercounter.com/characters-to-words>

(2024) demonstrated that GraphRAG outperforms traditional text-based RAG across diverse datasets, including Amazon financial reports, scientific texts on COVID-19 vaccines, technical specifications from aeronautics, and European environmental directives. The evaluation results showed that GraphRAG achieved an accuracy of 80% in providing correct answers, significantly surpassing the 50.83% accuracy attained by traditional RAG systems. However, we are unable to make any conclusions regarding the performance comparison between the KG-based and LLM-based QA systems due to the lack of available related work. Furthermore, the performance of KG-based systems often depends on the complexity of the KG and the query generation method. For instance, template-based approaches might offer high precision for more direct, clear questions but may struggle with handling more diverse or ambiguous questions. Conversely, LLM-based systems offer greater flexibility and can handle a wider range of natural language inputs, but they often require additional mechanisms such as fine-tuning or integration with external knowledge sources (like in RAG) to achieve competitive accuracy.

CHAPTER 3

METHODOLOGIES

In this chapter, we will explain the research methodology we employ, including the big picture of the experiment we conduct, the dataset and knowledge bases, as well as the evaluation metrics.

3.1 Research Methodology

The research methodology we employ is the development and application studies. Specifically, we are using the design science research (DSR) studies paradigm (vom Brocke et al., 2020). This paradigm aims to enhance human knowledge by developing innovative artifacts and generating design knowledge through creative solutions to real-world problems (Hevner et al., 2004). It comprises six main activities: problem identification and motivation, definition of the objectives for a solution, design and development, demonstration, evaluation, and communication. The following is the explanation of each activity, in relation to our work.

1. **Problem identification and motivation:** Naïve RAG systems face challenges with interpretability due to their reliance on opaque vector representations. RAG with KG as the knowledge base, i.e., GraphRAG can partially alleviate this problem. However, there are very few generic frameworks for GraphRAG that solely leverage open-source components, work for all KGs, and do not require training at all. Yet, this is crucial as it allows the users to customize the system according to their preferences while enhancing transparency and accessibility.
2. **Definition of the objectives for a solution:** Referring to the defined problem, our objective is to develop a RAG system with KG as the primary knowledge base using only open-source components. This framework will provide users with a transparent and flexible system that can be tailored to their specific needs without requiring additional training efforts.
3. **Design and development:** The GraphRAG architecture is designed using only open-source components. Following the design phase, we develop the actual artifact of the RAG system, ensuring it functions accordingly.
4. **Demonstration:** After developing the system, the functionality is demonstrated by

applying it to the real use case, i.e., QA over Wikidata, DBpedia, and local KG.

5. **Evaluation:** Quantitatively, the system’s performance in generating correct answers will be assessed by employing the Jaccard similarity measure. This metric will ensure that the system not only produces contextually appropriate responses but also adheres to the objective of utilizing only open-source components.
6. **Communication:** The architecture and implementation details of this pipeline are documented in a research report, such as this paper. Additionally, the experiment findings are also reported to ensure transparency and reproducibility.

3.2 Experiment Details

In general, we will be experimenting with two different versions of the pipeline, with the architecture of the first pipeline being simpler than the second one. Additionally, the scope of the first pipeline is also narrower. The details are as follows.

1. The first pipeline is used as the “prototype” and lays the foundation for building the second pipeline. As previously mentioned, this pipeline has a narrower scope, focusing specifically on Wikidata. The steps include the entities extraction and linking, SPARQL query generation, as well as answer generation. However, the property candidates are presented directly in the query generation prompt and obtained from the top-100 frequently used properties listed in Wikidata’s website¹. This highlights the limitation of the types of questions the system can support, as properties are only taken from this predefined list.
2. Departing from the limitation of the first pipeline, we refine the architecture by incorporating auxiliary vector databases to enable the usage of larger number of properties for the second pipeline. To enhance the performance for simple (single-hop) questions, we also include the text retrieval method using dense text embeddings encoded from verbalized triples (see Subsubsection 4.2.3.1). The steps for this pipeline are composed of the entities extraction and linking, retrieval, and answer generation. In the retrieval stage, the system initially attempts to find whether the question can be answered using the verbalized triples of the entity obtained in the first stage or not. If the system is unsure about the answer, the system proceeds with the SPARQL query generation, with the entities retrieved from the KG’s corresponding

¹https://www.wikidata.org/wiki/Wikidata:Database_reports/List_of_properties/Top100

API (if any) or vector database and properties retrieved from the vector database based on the user's question being passed as the prompt context.

Due to the more specific scope covered by the first pipeline, we designed our own dataset to be tested on it (see Subsubsection 3.5.1). We test this first pipeline on two LLMs: Mistral 7B and Mistral NeMo. We use the instruction fine-tuned version of both models.

On the other hand, to ensure the flexibility of our pipeline, there are three knowledge bases being plugged into the second pipeline: Wikidata, DBpedia, and local KG (see Subsection 3.3). The datasets we use are QALD-9-Plus (Perevalov et al., 2022) for Wikidata and DBpedia, and the synthetic dataset generated by our dataset generator for the local KG (see Subsubsection 3.5.2.2). The LLMs we use for the second pipeline are Mistral NeMo, LLaMA 3.1 8B, Qwen2.5 Coder 7B, and Qwen2.5 7B. Likewise, all models are the instruction fine-tuned versions.

Greedy decoding is employed for text generation using all models for standardization and reproducibility. For the first pipeline, the maximum number of new tokens allowed for generation is 1000, whereas the second one is 1500. The remaining parameters are left the same as the default ones.

Lastly, we conduct our evaluation locally using instances with GPU support that are rented on an hourly basis.

3.3 Knowledge Bases

In this subsection, we will explain the knowledge bases used in testing the system: Wikidata, DBpedia, and the local curriculum KG. We use Wikidata and DBpedia to represent open knowledge graphs, while the local curriculum KG serves as a representative of enterprise knowledge graphs.

3.3.1 Wikidata

Wikidata, originally launched in October 2012 by Wikimedia Foundation (Vrandečić and Krötzsch, 2014), is an openly accessible, collaborative knowledge base that is editable by anyone (Waagmeester et al., 2020). Consequently, the scope of knowledge covered by

Wikidata is nearly boundless, just like its sister project, Wikipedia². However, in contrast to Wikipedia which provides knowledge primarily in the free text format, Wikidata represents information in the structured format (Waagmeester et al., 2020). The core structure of Wikidata is made up of items, properties, and values, which are connected in statements that closely resemble RDF triples (Cantalops et al., 2019). Finally, the knowledge in Wikidata can be retrieved by SPARQL querying through Wikidata Query Service (WDQS)³, whose interface can be seen in Figure 3.1. The query example used in Figure 3.1 returns the URI of the instances (`wdt:P31`) of Cat (`wd:Q146`) alongside with its label. The Wikidata page of `wdt:P31` and `wd:Q146` can be seen in Figure 3.2 and Figure 3.3, respectively.

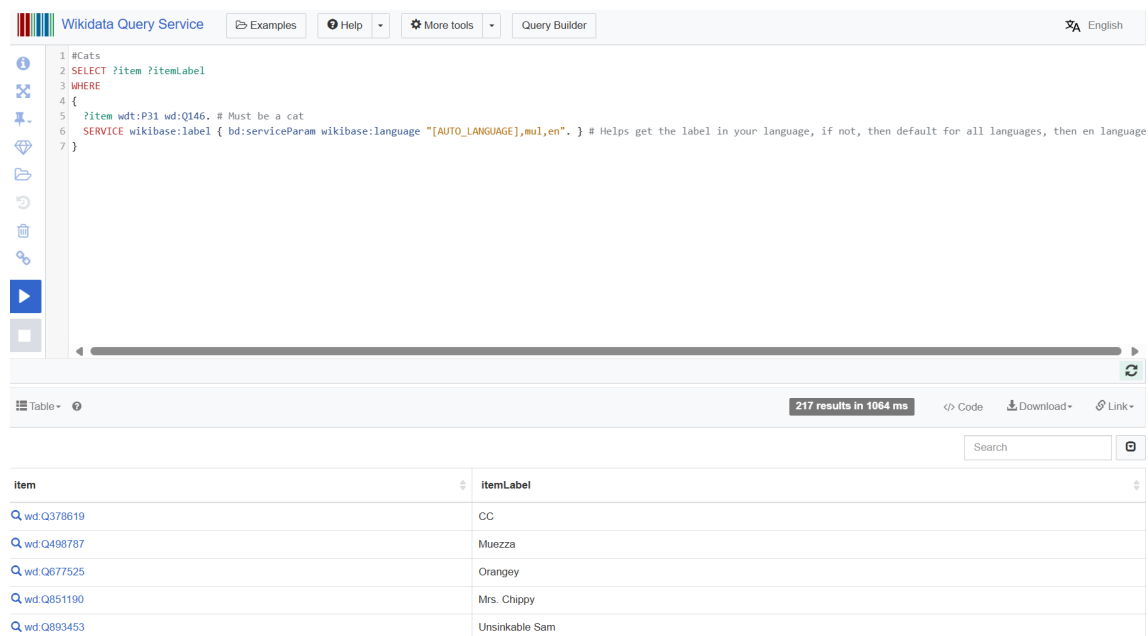


Figure 3.1: Wikidata Query Service's Interface

²<https://www.wikipedia.org/>

³<https://query.wikidata.org/>

Wikidata

English Not logged in Talk Contributions Create account Log in

Property Discussion Read View history Search Wikidata

instance of ^(P31)

that class of which this subject is a particular example and member; different from P279 (subclass of); for example: K2 is an instance of mountain; volcano is a subclass of mountain (and an instance of volcanic landform)

is a | is an | unique individual of | unitary element of class | rdf:type | type | ∈ | example of | is of type | has type | item of type

► In more languages

Data type

Item

Statements

instance of	Wikidata property
	▼ 0 references
	Wikidata property for the relationship of the element to its class
	▼ 0 references
	fundamental Wikidata entity

Figure 3.2: Wikidata Page of “instance of” (wdt:P31)

Wikidata

English Not logged in Talk Contributions Create account Log in

Item Discussion Read View history Search Wikidata

house cat ^(Q146)

domesticated feline

cat | domestic cat | housecat | puss | cats

► In more languages

Statements

instance of	organisms known by a particular common name
	▼ 0 references
subclass of	domesticated mammal
	start time 9. millennium BCE
	sourcing circumstances circa
	valid in place Fertile Crescent
	► 1 reference
	pet
	▼ 0 references

Figure 3.3: Wikidata Page of “house cat” (wd:Q146)

3.3.2 DBpedia

DBpedia project is an initiative to extract structured information on Wikipedia and making this information available in the Web (Bizer et al., 2009). This structured information in the form of open linked data (KG) is represented in RDF triples. Each entity in DBpedia

is identified by a URI in the form of `http://dbpedia.org/resource/name`, where **name** is obtained from the URL of the corresponding Wikipedia article, which is in the form of `http://en.wikipedia.org/wiki/name` (Auer et al., 2007). Similar to Wikidata, DBpedia also provides SPARQL endpoint⁴ to query the knowledge base, which is hosted on Virtuoso Universal Server⁵. Additionally, the ontology (schema) for DBpedia is available, created through crowdsourcing effort.

The interface of the DBpedia's SPARQL Query Editor to query the knowledge base through its endpoint API can be seen in Figure 3.4. The example query mentioned in Figure 3.4 returns the birth date (`dbo:birthDate`) of Lionel Messi (`dbr:Lionel_Messi`). Just like Wikidata, each resource in DBpedia has its own dedicated page as well. The page for `dbo:birthDate` can be seen in Figure 3.5 and `dbr:Lionel_Messi` in Figure 3.6.

Figure 3.4: DBpedia's SPARQL Query Editor Interface

⁴<https://dbpedia.org/sparql>

⁵<https://virtuoso.openlinksw.com/>

DBpedia Browse using Formats Faceted Browser Sparql Endpoint

About: [birth date](#)

An Entity of Type: [Property](#), from Named Graph: <http://dbpedia.org/resource/classes#>, within Data Space: [dbpedia.org](#)

Property	Value
rdf:type	owl:FunctionalProperty rdf:Property owl:DatatypeProperty
rdfs:domain	dbo:Animal
rdfs:isDefinedBy	http://dbpedia.org/ontology/
rdfs:label	birth date (en)
rdfs:range	xsd:date
owl:equivalentProperty	wikidata:P569 https://d-nb.info/standards/elementset/gnd#dateOfBirth schema:birthDate
wdrs:describedby	dbo:data/definitions.ttl
prov:wasDerivedFrom	http://mappings.dbpedia.org/index.php/OntologyProperty:birthDate

Figure 3.5: DBpedia Page of “birth date” (`dbo:birthDate`)

DBpedia Browse using Formats Faceted Browser Sparql Endpoint

About: [Lionel Messi](#)

An Entity of Type: [animal](#), from Named Graph: <http://dbpedia.org>, within Data Space: [dbpedia.org](#)

Lionel Andrés Messi (Spanish pronunciation: [ljoˈnel anˈdres ˈmesi]; born 24 June 1987), also known as Leo Messi, is an Argentine professional footballer who plays as a forward for Ligue 1 club Paris Saint-Germain and captains the Argentina national team. Widely regarded as one of the greatest players of all time, Messi has won a record seven Ballon d'Or awards, a record six European Golden Shoes, and in 2020 was named to the Ballon d'Or Dream Team. Until leaving the club in 2021, he had spent his entire professional career with Barcelona, where he won a club-record 35 trophies, including 10 La Liga titles, seven Copa del Rey titles and four UEFA Champions Leagues. A prolific goalscorer and creative playmaker, Messi holds the records for most goals in La Liga (474), most goals in a La L



Property	Value
dbo:Person/height	170.0
dbo:abstract	Lionel Andrés Messi (Spanish pronunciation: [ljoˈnel anˈdres ˈmesi]; born 24 June 1987), also known as Leo Messi, is an Argentine professional footballer who plays as a forward for Ligue 1 club Paris Saint-Germain and captains the Argentina national team. Widely regarded as one of the greatest players of all time, Messi has won a record seven Ballon d'Or awards, a record six European Golden Shoes, and in 2020 was named to the Ballon d'Or Dream Team. Until leaving the club in 2021, he had spent his entire professional career with Barcelona, where he won a club-record 35 trophies, including 10 La Liga titles, seven Copa del Rey titles and four UEFA Champions Leagues. A prolific goalscorer and creative playmaker, Messi holds the records for most goals in La Liga (474), most goals in a La Liga

Figure 3.6: DBpedia Page of “Lionel Messi” (`dbo:Lionel_Messi`)

3.3.3 Local KGs

For the local KG, we leverage Fasilkom UI’s Computer Science Major curriculum knowledge graph built by Mandolang (2024) as hosted on GitHub. This KG is extracted from the Fasilkom UI’s 2024 Curriculum Guidebook for Computer Science Major, which

is publicly available through Fasilkom UI's LMS⁶. Specifically, the content is obtained from page 122 to 186, covering the syllabus of all courses. Additionally, some auxiliary information is manually added by the creator. The retrieved content is then converted into KG with the help of GLiNER (Zaratiana et al., 2023) to extract entities from the unstructured text data.

The constructed KG is saved in the form of triples and exported into a txt file. In total, there are 1628 triples, containing the information of courses. The number of properties available in the dataset is 10, which includes kode mata kuliah, kategori mata kuliah, disebut juga, prasyarat, bobot SKS, metode evaluasi, lab, deskripsi, topik, and cpmk. Some triples about the “Basis Data” course can be seen in Code 3.1.

```

1 ('Basis Data', 'kode mata kuliah', 'CSGE602070')
2 ('Basis Data', 'kategori mata kuliah', 'Mata Kuliah Wajib Fakultas')
3 ('Basis Data', 'disebut juga', 'basdat')
4 ('Basis Data', 'disebut juga', 'Database')
5 ('Basis Data', 'disebut juga', 'DB')
6 ('Basis Data', 'disebut juga', 'BD')
7 ('Basis Data', 'prasyarat', 'Dasar-dasar Pemrograman 2')
8 ('Basis Data', 'bobot SKS', '4')
9 ('Basis Data', 'metode evaluasi', 'Ujian')
10 ('Basis Data', 'metode evaluasi', 'Proyek Kelompok')
11 ('Basis Data', 'lab', 'WorkshopLab')
12 ('Basis Data', 'deskripsi', 'Kuliah ini membahas konsep dasar pengelolaan basis data,
13     meliputi aspek pemodelan dan perancangan, bahasa, fasilitas, implementasi, dan
14     aplikasi dari basis data.')
15 ('Basis Data', 'topik', 'Concept and Architecture of Database System')
16 ('Basis Data', 'topik', 'Modelling using Entity Relationship')
17 ('Basis Data', 'topik', 'Relational DB design; ER/EER to relational mapping')
18 ('Basis Data', 'topik', 'Relational data model & relational DB constraint')
19 ('Basis Data', 'topik', 'SQL-DDL Insert/Update/Delete')
20 ('Basis Data', 'topik', 'SQL Basic & Advanced Query')
21 ('Basis Data', 'topik', 'SQL - Indexing, Trigger and Stored Procedure')
22 ('Basis Data', 'topik', 'Functional Dep. and Normalization')
23 ('Basis Data', 'cpmk', 'Mampu merancang, implementasi, dan mengelola basis data sesuai
24     kebutuhan')

```

Code 3.1: Triples about “Basis Data” Course

We convert the “raw” triples into RDF triples, since our system only supports SPARQL for query generation, with the total of 874 triples. Prior to this, each name is translated into English, except for the value of disebut juga property. We define the prefix to

⁶https://scele.cs.ui.ac.id/pluginfile.php/1279/block_html/content/buku_panduan_kur_2024_ilmu_komputer.pdf

be `ns1:`, representing `http://example.org/`. The mapping of each property—with some exclusions on properties we think are unnecessary⁷—is defined in Table 3.1. Additionally, we define `ns1:course`, `ns1:evaluation`, `ns1:course_category`, and `ns1:research_lab` as the classes. A more detailed ontology definition is described in Subsubsection 3.4.3.

Original Dataset	RDF Dataset
kode mata kuliah	<code>ns1:has_course_code</code>
kategori mata kuliah	<code>ns1:has_course_category</code>
disebut juga	<code>ns1:also_known_as</code>
prasyarat	<code>ns1:has_prerequisite</code> <code>ns1:has_prerequisite_course</code> <code>ns1:has_minimum_credits</code>
bobot SKS	<code>ns1:has_credits</code>
metode evaluasi	<code>ns1:has_evaluation_method</code>

Table 3.1: Properties Mapping

The triples in Code 3.1 if converted into RDF triples would result in the following triples, as in Code 3.2.

```

1 @prefix ns1: <http://example.org/> .
2 @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
3 @prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
4
5 ns1:advanced_database a ns1:course ;
6   rdfs:label "Advanced Database" ;
7   ns1:also_known_as "Advanced Database",
8     "banjut" ;
9   ns1:has_course_category ns1:study_program_elective_course ;
10  ns1:has_course_code "CSCE604174" ;
11  ns1:has_credits 3 ;
12  ns1:has_evaluation_method ns1:lab_assignment ;
13  ns1:has_prerequisite_course ns1:database ;
14  ns1:has_research_group ns1:reliable_software_engineering .

```

Code 3.2: RDF Version of Code 3.1

⁷The objects of such properties are not “neat” enough to be either entity or literal.

3.4 Ontology

The ontology here refers to the properties and classes defined in the KG. In this section, we will delve into the ontology details for each KG.

3.4.1 Wikidata

For Wikidata, we obtain all properties that exist in the KG using its SPARQL endpoint. Specifically, the URI of each property and its human-readable label are retrieved to be provided as context using hybrid search. However, since the classes are searchable through Wikidata API (along with entities), we do not include them in this section.

3.4.2 DBpedia

For DBpedia, we use the provided DBpedia's T-Box Ontology⁸. This is an OWL file containing information on classes, object properties, and data properties that exist in DBpedia. However, the available information is limited to properties and classes whose prefix started with `dbo:` or `http://dbpedia.org/ontology/`. For each property or class, the human-readable label is retrieved for property and class linking purposes as in Subsubsection 4.2.3.2 using hybrid search. Additionally, we also obtain the domain and range of each property (if applicable) which we include in the SPARQL query generation prompt for additional context.

3.4.3 Local KGs

For the local curriculum KG as described in Subsubsection 3.3.3, the properties and classes are obtained directly from the RDF graph. The complete ontology we defined is as follows in Table 3.2.

⁸<https://www.dbpedia.org/resources/ontology/>

ns1:also_known_as	rdfs:domain	ns1:course OR ns1:research_lab
ns1:also_known_as	rdfs:range	xsd:string
ns1:has_course_category	rdfs:domain	ns1:course
ns1:has_course_category	rdfs:range	ns1:course_category
ns1:has_course_code	rdfs:domain	ns1:course
ns1:has_course_code	rdfs:range	xsd:string
ns1:has_credits	rdfs:domain	ns1:course
ns1:has_credits	rdfs:range	xsd:integer
ns1:has_evaluation_method	rdfs:domain	ns1:course
ns1:has_evaluation_method	rdfs:range	ns1:evaluation
ns1:has_minimum_credits	rdfs:domain	ns1:course
ns1:has_minimum_credits	rdfs:range	xsd:integer
ns1:has_prerequisite_course	rdfs:domain	ns1:course
ns1:has_prerequisite_course	rdfs:range	ns1:course
ns1:has_research_group	rdfs:domain	ns1:course
ns1:has_research_group	rdfs:range	ns1:research_lab
ns1:has_minimum_credits	rdfs:subPropertyOf	ns1:has_prerequisite
ns1:has_prerequisite_course	rdfs:subPropertyOf	ns1:has_prerequisite

Table 3.2: Ontology of the Local Curriculum KG

3.5 Dataset

We will explain the datasets we leverage to assess the performance of our proposed system.

3.5.1 Pipeline v1 Dataset

Since the first pipeline is exclusively for Wikidata, we hand-crafted the dataset for evaluation from Wikidata as retrieved on August 28th, 2024. This was done to ensure that the dataset complies with the requirements, such as the used properties must be in the top-100 list and the SPARQL queries are designed to return a small number of rows. Additionally, this strategy helps maintain the output within the manageable limit of the LLM's context window. The questions in this evaluation dataset are also distinct from those used during few-shot learning to prevent any potential data leakage.

The dataset consists of 31 rows encompassing three main domains (Wikidata classes): animal, artist, and written work. Specifically, there are 8 rows within the animal domain, 11 rows within the artist domain, and 12 rows within the written work domain. Some examples of the data are shown in Table 3.3.

Query Name	SPARQL Query	Category
List of Mammals by Name	SELECT ?mammal ?mammalLabel WHERE { ?mammal wdt:P31 wd:Q7377 . SERVICE wikibase:label { bd:serviceParam wikibase:language 'en' . } }}	Animal
Date of birth of Leonardo da Vinci	SELECT ?date_of_birth WHERE { wd:Q762 wdt:P569 ?date_of_birth .}	Artist
Number of works authored by J. R. R. Tolkien	SELECT (COUNT(*) AS ?count) WHERE { ?work wdt:P50 wd:Q892 .}	Written Work

Table 3.3: Sample Queries from Dataset v1

3.5.2 Pipeline v2 Dataset

For the second pipeline, we leverage these datasets.

3.5.2.1 Wikidata and DBpedia

For Wikidata and DBpedia, we leverage the QALD-9-Plus (Perevalov et al., 2022) dataset. QALD-9-Plus is a multilingual KGQA benchmark dataset for Wikidata and DBpedia. It covers 10 languages: English, German, Russian, French, Spanish, Armenian, Belarusian, Lithuanian, Bashkir, and Ukrainian. The dataset consists of two splits: train split (408 unique questions) and test split (150 unique questions). However, in this work, we will only use the former split in the English language.

Following our research scope and the obtained ontology information for DBpedia (which only supports properties and classes with the `dbo:` prefix), we filter out dataset entries whose queries incorporate resources that use other than `dbo:` prefix and return either resource or number only. We also ensure that each entry we take for DBpedia also has the equivalent Wikidata query.

Upon filtering irrelevant entries, we downsample the filtered set to 14 entries for in-context learning in SPARQL generation (Subsubsection 4.2.3.2) and 65 entries for the evaluation of the entire system. We leverage the cosine similarity downsampling technique to obtain the subset of entries that are the least relevant to the others to ensure each “cluster” has at least a representative in the subset. The code used for this downsampling is available in Code 3.3. In general, the steps taken are as follows.

1. Encode all questions into `all-mpnet-base-v2` version of MPNet embeddings.
2. Compute pairwise cosine similarities for each question.
3. Average the cosine similarity values for each question.
4. Sort the question based on the average cosine similarity in ascending order.
5. Select the top- k questions with the lowest average cosine similarity. In this case, we have obtained the k questions that are the least similar to the rest.

```

1 import pandas as pd
2 import numpy as np
3 from sentence_transformers import SentenceTransformer
4 from sklearn.metrics.pairwise import cosine_similarity
5
6 model = SentenceTransformer('all-mpnet-base-v2')
7
8 def cosine_downsampling(answertype, downsample_size=100):
9     qald_dbpedia_nonagg_df = qald_dbpedia_df[
10         (qald_dbpedia_df.aggregation == False) &
11         (qald_dbpedia_df.answertype == answertype)
12     ]
13     texts = qald_dbpedia_nonagg_df["question"].tolist()
14
15     embeddings = model.encode(texts, show_progress_bar=True)
16     similarities = cosine_similarity(embeddings)
17     pairwise = np.mean(similarities, axis = 1)
18     qald_dbpedia_nonagg_df.loc[:, "pairwise"] = pairwise
19     qald_dbpedia_nonagg_df.sort_values("pairwise", ascending = True, inplace=True)
20     qald_dbpedia_nonagg_df = qald_dbpedia_nonagg_df
21         .head(downsample_size)
22         .reset_index(drop=True).copy()
23     return qald_dbpedia_nonagg_df

```

Code 3.3: Python Code for Dataset Downsampling

Some examples of downsampled data can be seen in the following Table 3.4.

Query Name	Wikidata Query	DBpedia Query
Give me all movies with Tom Cruise.	SELECT DISTINCT ?uri WHERE { ?uri wdt:P161 wd:Q37079. }	SELECT DISTINCT ?uri WHERE { ?uri a dbo:Film ; dbo:starring dbr:Tom_Cruise }
In which countries can you pay using the West African CFA franc?	SELECT DISTINCT ?uri WHERE { ?uri wdt:P38 wd:Q861690 . }	SELECT DISTINCT ?uri WHERE { ?uri dbo:currency dbr:West_African_CFA_franc } }
How much did Pulp Fiction cost?	SELECT DISTINCT ?value WHERE { wd:Q104123 wdt:P2130 ?value . }	SELECT DISTINCT ?n WHERE { dbr:Pulp_Fiction dbo:budget ?n } }
Which countries have more than two official languages?	SELECT DISTINCT ?uri WHERE { ?uri wdt:P31 wd:Q6256 . ?uri wdt:P37 ?language . } GROUP BY ?uri HAVING (COUNT(?language)>2)	SELECT DISTINCT ?uri WHERE { ?uri a dbo:Country ; dbo:officialLanguage ?language } GROUP BY ?uri HAVING (COUNT(?language) > 2)
How many languages are spoken in Turkmenistan?	SELECT (COUNT(DISTINCT ?x) AS ?c) WHERE { dbr:Turkmenistan dbo:language ?x } }	SELECT (COUNT(DISTINCT ?uri) as ?c) WHERE { wd:Q874 wdt:P2936 ?uri . }

Table 3.4: Sample Queries from Dataset v2 for Wikidata and DBpedia (QALD)

To evaluate the multilingual capabilities of the system, we also translated the English dataset into Indonesian using the GPT-4 model of ChatGPT. This translation allows us to test the system’s performance in processing non-English queries and ensures broader applicability to diverse linguistic contexts.

3.5.2.2 Local KGs

Given this framework’s KG-agnostic nature, we also provide an automatic dataset generation system which data can be used for training and testing, especially for local KGs. This system supports the generation of four types of questions, as described in

Table 3.5. Additionally, queries with `COUNT` clause can also be generated, leveraging the same triple patterns. The output of this system is a dataset that consists of three columns: the question in natural language, the appropriate SPARQL query, and the category of the question. Below are the five main steps required to generate one question.

1. **Entity selection.** This step involves randomly selecting one entity (i.e., the instance of a class) from the given KG.
2. **Random walk.** This step involves performing a random walk on the KG based on the category of the question category, starting from the node selected in the previous step. We also provide a facility if the users would like to exclude some properties from being used in the generated question. For instance, schematic properties like `rdfs:domain` and `rdfs:range` can be excluded if the users intend to focus more on the concrete, data graph.
3. **Resource label resolver.** This step involves resolving the natural language label for each property and entity found in the triples.
4. **SPARQL query formation.** This step involves taking the triples retrieved in the second step and encompassing them into one SPARQL query using a pre-defined template, based on the question category.
5. **Question generation.** This step involves generating the question in natural language based on the generated SPARQL query and the resources' labels using LLM. The generation prompt leverages the zero-shot prompt template defined by Both and Perevalov (2024) with off-the-shelf instructions fine-tuned LLM.

Category	Triple Pattern
Simple 1	{ s p ?o . }
Simple 2	{ ?s p o . }
Complex 1	{ ?s p1 o1; p2 o2 . }
Complex 2	{ ?s p1 ?o1 . ?o1 p2 o2 . }

Table 3.5: Supported Question Categories

For the local curriculum KG, the proportion of each category for the test set includes 9 simple 1 questions, 8 simple 2 questions, 5 complex 1 questions, 5 complex 2 questions,

5 simple 1 COUNT questions, and 5 simple 2 COUNT questions. Meanwhile, the data in the train set (few-shot) composed of 2 simple 1 questions, 2 simple 2 questions, a complex 1 question, a complex 2 question, a simple 1 COUNT question, and a simple 2 COUNT question. The test set and train set are disjoint to prevent data leakage.

We leverage Qwen/Qwen2.5-3B-Instruct for the verbalization process, i.e., converting the SPARQL query into the respective natural language question. The prompt used for generation can be seen below.

SPARQL Verbalization to Natural Language Question

Having a SPARQL query:

```
select ?x { <http://example.org/machine_learning
<http://example.org/has_prerequisite_course> ?x . }
```

Where:

http://example.org/machine_learning has human-readable name 'Machine Learning'

http://example.org/has_prerequisite_course has human-readable name 'has prerequisite course'

Transform the SPARQL query to a natural language question.

Output just the transformed question

Upon generating the set of questions, we realize that the quality of the generated sentences is quite low, due to the lack of context provided to the LLM. To ensure that the intent of the question is conveyed clearly, we refine some ambiguous questions generated by the LLM. Examples of some ready-to-use questions are mentioned in Table 3.6.

Question	Category	SPARQL Query
What is the course category of 'Spoken Language Processing'?	Simple 1	<pre>select ?x { ns1:spoken_language_processing ns1:has_course_category ?x . }</pre>
What courses that have 3 credits?	Simple 2	<pre>select ?x { ?x ns1:has_credits '3'^^xsd:integer . }</pre>
What courses have 'Group Project' and 'Test' as evaluation methods and have the course code 'CSGE603130'?	Complex 1	<pre>select ?x { ?x ns1:has_evaluation_method ns1:group_project . ?x ns1:has_evaluation_method ns1:test . ?x ns1:has_course_code 'CSGE603130' . }</pre>
What courses have study program mandatory courses as their prerequisites?	Complex 2	<pre>select ?x { ?x ns1:has_prerequisite_course ?y . ?y ns1:has_course_category ns1:study_program_mandatory_course . }</pre>
How many courses are prerequisites for Calculus 2?	COUNT Simple 1	<pre>select (count(?x) as ?cnt) { ns1:calculus_2 ns1:has_prerequisite_course ?x . }</pre>
How many courses are categorized as 'Faculty Mandatory Course'?	COUNT Simple 2	<pre>select (count(?x) as ?cnt) { ?x ns1:has_course_category ns1:faculty_mandatory_course . }</pre>

Table 3.6: Sample Questions for the Local Curriculum KG

3.6 Evaluation Metric

We assess the performance of our system by comparing the output of Subsubsection 4.1.4 and Subsubsection 4.2.3, i.e., the generated SPARQL query execution results and the ground truth SPARQL query execution results as provided in the dataset. Even though the component for generating natural language is provided within the pipeline, we do not evaluate the generated natural language text for these two reasons:

1. The dataset we use for evaluating the second pipeline for open KGs, QALD, only provides the ground truth in the form of SPARQL query and its execution result. It does not provide the natural language text, whereas using LLM to perform verbalization over the ground truth set might introduce bias and discrepancies between the result set and its natural language text due to LLM's hallucination phenomenon. Notably, the QALD 9 dataset itself is specifically designed to benchmark systems that return the correct set of answers or generate a SPARQL query to retrieve those answers, not the natural language text (Usbeck et al., 2018).
2. We reckon the retrieved result set as the main reason for the correctness of answers generated by this particular RAG system, since the generator (LLM) only wraps the values in the retrieved set in the form of natural language. This is in contrast to naïve RAG which requires LLM's inherent knowledge to extract specific information from the retrieved document chunk in order to answer the user's question.

An automatic tool for evaluating the answers of QA over KG using this query execution result scheme, GERBIL QA (Verborgh et al., 2019), is commonly used for systems that leverage QALD as the benchmark dataset. In this system, the set of URIs or literals generated by the system is compared to the ground truth set, and the traditional precision, recall, and F-measure are used for the evaluation. To be specific, for each set of answers, if the set of returned resources by the QA system intersects with the ground truth set, the answer is considered correct, i.e., counted as a true positive. However, the limitation of such evaluation schema is that it primarily focuses on the intersection between the ground truth and generated sets only, without adequately penalizing for irrelevant results or fully considering the completeness of the retrieved answers.

To remedy the limitation in GERBIL QA, different evaluation schema was introduced. Usbeck et al. (2018) proposed using recall and precision to evaluate the correctness of the

answer sets for each question q , as in Equation 3.1 and 3.2, respectively.

$$\text{recall}(q) = \frac{\text{number of correct system answers for } q}{\text{number of gold standard answers for } q} \quad (3.1)$$

$$\text{precision}(q) = \frac{\text{number of correct system answers for } q}{\text{number of system answers for } q} \quad (3.2)$$

This evaluation schema further computes the micro and macro F-measure of the system over all test questions. The micro F-measure is calculated by summing up all the true and false positives and negatives, and compute the precision, recall, and F-measure at the end. However, for the macro F-measure, precision, recall, and F-measure is calculated individually for each test question, then the final result is obtained by averaging those values for each metric.

On the other hand, we propose using Jaccard Similarity to evaluate the result of our system. Jaccard Similarity for two sets A and B is defined as the number of elements in the intersection of A and B divided by the number of elements in their union, mathematically expressed as in the following Equation 3.3.

$$\text{Jaccard}(A, B) = \frac{|A \cap B|}{|A \cup B|} \quad (3.3)$$

By using Jaccard Similarity, the comparison between the generated and ground truth sets of answers is quantified as a single similarity score, representing the overall degree of overlap between the two sets. While precision and recall offer more granular insights into the system's accuracy by measuring the proportion of relevant items retrieved (precision) and the proportion of retrieved items that are relevant (recall), they focus only on specific subsets (retrieved/relevant). As a result, they may overlook the holistic alignment between the two sets. In contrast, Jaccard Similarity explicitly measures the proportion of overlapping items between the predicted and ground truth sets, providing a more holistic and intuitive approach to interpreting retrieval results. This makes it easier to understand as a direct comparison of similarity.

However, this metric can be very sensitive to the size of the result sets. As the union of the predicted and ground truth sets grows larger—particularly when many irrelevant results are included in the predicted set—the score can decrease significantly, even when the

intersection contains many relevant answers. Consequently, this metric can exhibit bias on large datasets, especially when the predicted set is disproportionately large or contains many irrelevant results. In such cases, Jaccard Similarity may lead to a lower score that does not fully reflect the system's ability to retrieve relevant answers.

For the implementation matter, since the original SPARQL query execution results take the ordering of the rows into account to accommodate the `ORDER BY` clause, we have to transform both predicted and ground truth execution results into sets prior to applying this metric. Additionally, for further standardization, each row is treated as a tuple whose elements are ordered alphabetically. Thus, the A and B in Equation 3.3 are sets of tuples. The detailed illustration of this evaluation schema can be seen in Figure 3.7.

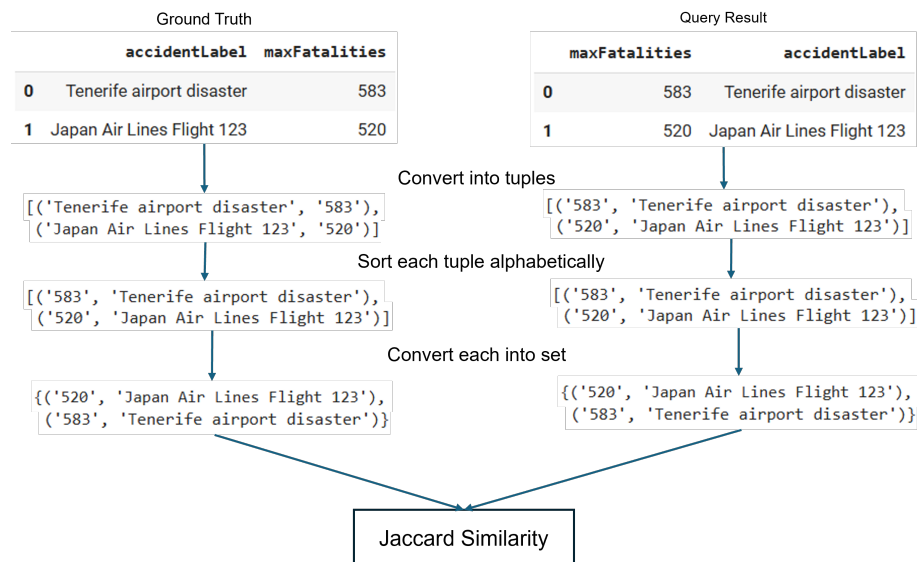


Figure 3.7: Evaluation Schema

CHAPTER 4

IMPLEMENTATION

In this chapter, we will explain the detailed implementation of the pipeline and the libraries or resources we utilize.

4.1 Pipeline v1

The initial pipeline is comprised of several stages, with a focus on QA over Wikidata KG only. This pipeline relies solely on generative LLM in determining appropriate resources and constructing SPARQL queries through in-context learning, i.e., a paradigm that allows language models to learn how to accomplish downstream tasks based on a few given examples (Brown et al., 2020). The pipeline is illustrated in Figure 4.1 and will be explained in detail subsequently.

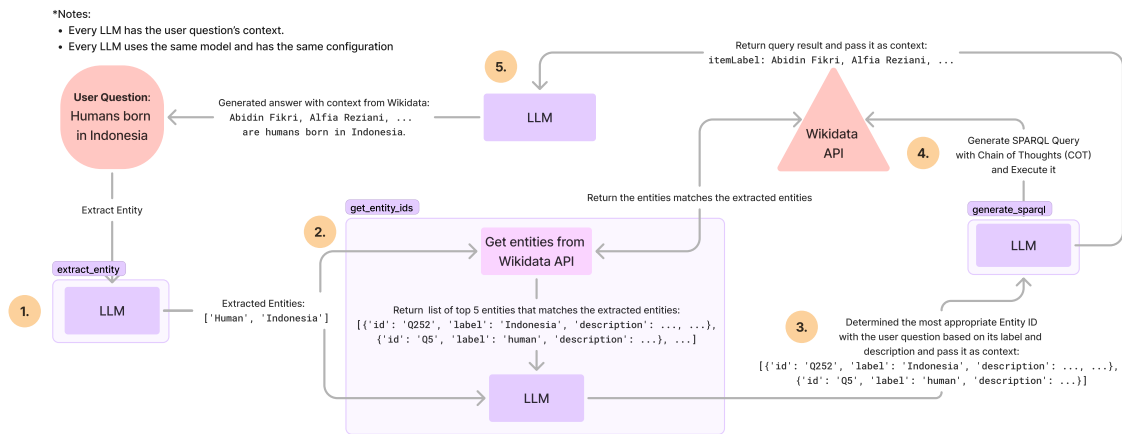


Figure 4.1: The Initial Pipeline

4.1.1 Entities Extraction

Given the user's question, the LLM identifies key entities mentioned within it. These entities serve as the “anchors” for generating the SPARQL query, which is subsequently used to query the KG. In this stage, we employ few-shot prompting by providing the model with 11 handcrafted input-output examples. For instance, if the question is about Humans born in Indonesia, the LLM response is expected to be ['Human', 'Indonesia']. The full prompt used for this task is shown below; however, only 3 of the 11 input-output pairs are displayed for brevity.

Entities Extraction Prompt

INSTRUCTIONS

- Extract the entities from the given question!
- These entities usage is to find the most appropriate entity ID from wikidata to be used in SPARQL queries.
- If there is no entity in the question, return empty list.
- ONLY return the entities. DO NOT return anything else.
- DO NOT include adjectives like 'Highest', 'Lowest', 'Biggest', etc in the entity.
- DO NOT provide any extra information, for instance explanation inside a brackets like '(population)', '(area)', '(place)', '(artist)', etc
- DO NOT include any explanations or apologies in your responses.
- Remove all stop words, including conjunctions like 'and' and prepositions like 'in' and 'on' from the extracted entity.
- Make the entity singular, not plural. For instance, if the entity is foods, then transform it into food.

OUTPUT FORMAT INSTRUCTIONS

{format_instructions}

EXAMPLES

- **Question:** how much is 1 tablespoon of water?

Entity: ```json>{"entities": ["Tablespoon"]}```

- **Question:** how are glacier caves formed?

Entity: ```json>{"entities": ["Glacier cave"]}```

- **Question:** how much are the harry potter movies worth?

Entity: ```json>{"entities": ["Harry Potter"]}```

...

QUESTION

- **Question:** {question}

Entity:

4.1.2 Entity ID Retrieval

For every extracted entity obtained in the preceding step, the system conducts a keyword-based search using the Wikidata API¹, in which the top-5 matches are selected. The retrieved data includes the details below.

- **URI (ID):** The unique identifier of the entity in Wikidata.
- **Label:** The human-readable label of the entity.
- **Description:** A brief description or context about the entity.

4.1.3 Entity Disambiguation and Selection

In this stage, we utilize LLM to determine the most appropriate entity based on the context of the user's question. This stage relies on the LLM's capability to interpret the intent of the question and the description of the retrieved entities. This could narrow down the options of a particular entity to the most relevant ones, ensuring the accuracy of the generated SPARQL queries. Just like in the first stage, few-shot prompting is utilized by providing 4 input-output pairs. The prompt used for this stage is shown below, with only 1 example being provided out of 4.

Entity Selection Prompt

INSTRUCTIONS

- For each entity given, find the most appropriate entity ID from the list of wikidata entities given to be used in SPARQL queries to answer the given question!
- You MUST return the ID and label of all the entity in the list of entities given in "Entities". For example: if the entities give is "A" and "B", then you have to return the ID and label of "A" and "B".
- ONLY return ONE set of entity ID for each entity, DO NOT hallucinate and pick more than one entity ID sets. For example: there are [C, D, E] for "A" and there are [F, G, H] for "B", ONLY return ONE for each entity, such as C for "A" and F for "B".
- ONLY return the entity IDs, labels, and descriptions from the list

¹<https://www.wikidata.org/w/api.php>

of wikidata entities given. DO NOT return anything else and DO NOT hallucinate.

- DO NOT include any explanations or apologies in your responses.

OUTPUT FORMAT INSTRUCTIONS

{format_instructions}

EXAMPLES

- **Question:** Horses

Entities: ["Horse"]

Wikidata Entities: ```json{"Horse": [{"id": "Q726",
"label": "horse",
"description": "domesticated four-footed mammal from the equine
family"}],
{"id": "Q60168",
"label": "heroin",
"description": "chemical compound; opioid most commonly used as a
recreational drug for its euphoric effects"}],
{"id": "Q136",
"label": "knight",
"description":
"piece in the board game of chess"}],
{"id": "Q10758650",
"label": "Equus caballus",
"description": "species of mammal"}],
{"id": "Q840330",
"label": "pommel horse",
"description": "men's artistic gymnastics apparatus"}]}```

Entity IDs: ```json{
"ids": [
{"id": "Q726", "label": "horse", "description": "domesticated
four-footed mammal from the equine family"}
]
}```

```
...

## QUESTION
- Question: {question}
Entities: {entities}
Wikidata Entities: ```json{retrieved.wikidata.matched.entities}```
Entity IDs:
```

4.1.4 SPARQL Query Generation

Given the specific entities obtained from the previous stage, the LLM will generate the corresponding SPARQL query to answer the user's question. We provide a list of top-100 frequently used properties as a reference to generate the SPARQL query, obtained from Wikidata's website². The SPARQL query must contain the given entities and properties only to reduce hallucination in the generation. Additionally, to enhance the generation, we leverage chain-of-thought prompting, which enables complex reasoning through a series of intermediate reasoning steps (Wei et al., 2024a). We further combine this with few-shot prompting by giving 9 input-output pairs as examples, with the chain of thoughts generated by ChatGPT³, specifically the GPT-4o model. The prompt can be seen below, with only several properties and one input-output pair example being provided.

SPARQL Query Generation Prompt

```
## INSTRUCTIONS
- Generate SPARQL queries to answer the given question!
- To generate the SPARQL, you can utilize the information from the
  given Entity IDs. You do not have to use it, but if it can help you
  to determine the ID of the entity, you can use it.
- You will also be provided with the 100 most used properties with its
  ID. You are only able to generate SPARQL query from these properties. If
  it requires property that is not provided, then generate empty query like
  ```sparql```.
- You can also determine the IDs of the entities that aren't provided with
```

<sup>2</sup>[https://www.wikidata.org/wiki/Wikidata:Database-reports/List\\_of\\_properties/Top100](https://www.wikidata.org/wiki/Wikidata:Database-reports/List_of_properties/Top100)

<sup>3</sup><https://chatgpt.com/>

your knowledge.

- Generate the SPARQL with chain of thoughts.
- DO NOT include any apologies in your responses.
- ONLY generate the Thoughts and SPARQL query once! DO NOT try to generate the Question!
- When using a property such as P17 (country), you DO NOT need to verify explicitly whether it is Q6256 entity (country).
- DO NOT use LIMIT, ORDER BY, FILTER in the SPARQL query when not explicitly asked in the question!
- DO NOT use aggregation function like COUNT, AVG, etc in the SPARQL query when not asked in the question!
- Always use 'en' language for labels as default unless explicitly asked to use another language.
- Be sure to generate a SPARQL query that is valid and return all the asked information in the question.
- Make the query as simple as possible!
- DO NOT hallucinate the thoughts and query!

## ## CONTEXT

- entity IDs: ``{entity\_ids}```
- 100 most used properties with its ID:  

```
```json [
{{
"label": "cites work",
"id": "P2860",
"description": "citation from one creative or scholarly work to another",
"aliases": "bibliographic citation, citation"
}},
{{
"label": "series ordinal",
"id": "P1545",
"description": "position of an item in its parent series (most frequently a 1-based index), generally to be used as a qualifier (different from "rank" defined as a class, and from "ranking" defined as a property for evaluating a quality).",
```

```
"aliases": "#, index, number, rank, n\u00b0, n\u00ba, \u2116, num.,
ordinal, ordinal number, position in series, section number, series
number, sort order, sorting order, unit number"
}},
...
]
``
```

EXAMPLES

- **Question:** Cats

Thoughts:

1. The question asks for information about cats, so I need to identify the relevant entities and properties in Wikidata.
2. First, I need to find items that are classified as cats. In Wikidata, "cat" corresponds to the entity with the identifier Q146.
3. To retrieve items that are instances of cats, I will use the property P31, which stands for "instance of."
4. I should also retrieve the labels of these items in a language the user understands. To do this, I'll utilize the SERVICE wikibase:label to get the label in the user's preferred language. If that language is unavailable, I'll default to a multilingual or English label.

SPARQL Query: ```sparql

```
SELECT ?item ?itemLabel
WHERE {{
  ?item wdt:P31 wd:Q146.
  SERVICE wikibase:label {{ bd:serviceParam wikibase:language
    "[AUTO_LANGUAGE],mul,en". }} # Helps get the label in your language,
    if not, then default for all languages, then en language
  }}
``
...

```

QUESTION

- **Question:** {question}

4.1.5 Query Execution and Answer Generation

The generated SPARQL query from the preceding stage is then executed against the Wikidata Query Service (WDQS) endpoint to fetch the query results, in the form of JSON. Upon obtaining the data, it will be augmented into the prompt to instruct the LLM to generate a relevant answer based on the user's question. We use a zero-shot prompting technique here, which means there is no input-output pair provided as a guide. By not constraining the LLM to generate a specific style of answer, we allow it to draw upon its pre-trained knowledge and contextual understanding to address a wide variety of potential questions posed by the user. The utilized prompt can be seen below.

Answer Generation Prompt

INSTRUCTIONS

- You are a master of Wikidata, you know everything about Wikidata because you are given the answer in the context.
- Generate the answer from the given question by utilizing the given context from Wikidata.
- Try your best to use the given context as the answer!
- DO NOT hallucinate and only provide answers from the given context.
- DO NOT make up an answer
- If you don't know the answer, just say that you don't know
- Answer the question in a natural way like you are the one who know the context, DO NOT mention like "according to the context", etc.
- Answer it using complete sentence!
- If the question is about retrieving information that is limited to a certain amount, make sure to return all the results from the context that match the limited amount.

CONTEXT

```
```json{{
"wikidata_response": {wikidata_context}
}}```
```

###### ## QUESTION

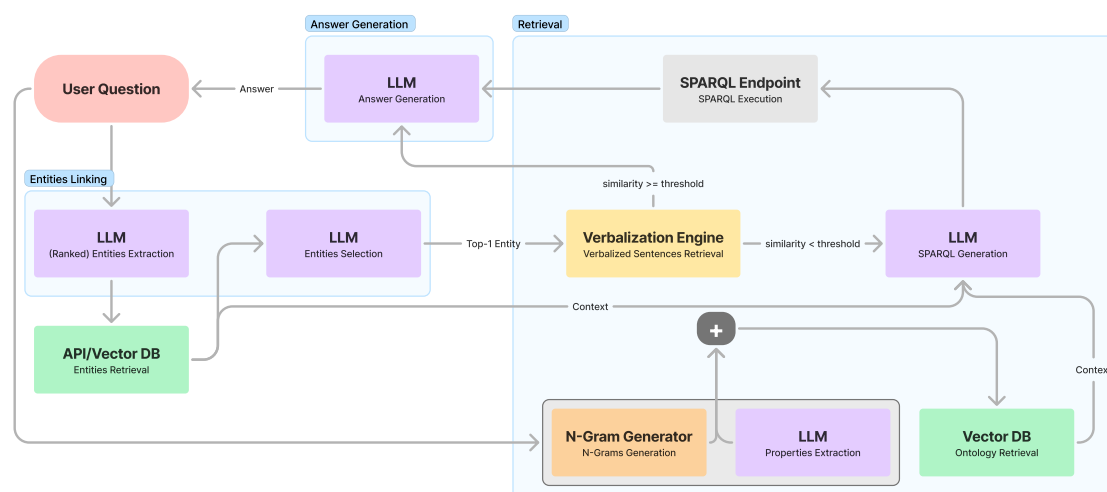
```
{question}
```

**## ANSWER**

## 4.2 Pipeline v2

We recognize the limitation of the initial pipeline, which restricts the usage of the properties to those only defined in the prompt. While this approach can help minimize the introduction of other external components, it lacks scalability. This raises a significant problem for large-scale KGs, where the number of properties can be thousands. Including all of these properties in the prompt explicitly is not possible due to the context window limitation and the weak performance of LLMs when dealing with lengthy prompts (i.e., long context scenarios) (Jiang et al., 2024).

In this upgraded version, we enable multilingual support and generalize the notion of the initial pipeline to accommodate all types of KGs, including large-scale ones, by introducing an external vector database component to select a relevant subset of properties needed for generating the accurate SPARQL query. This approach avoids the need to include the entire set of properties in the prompt explicitly. Furthermore, we incorporate several auxiliary steps aimed at improving the accuracy of the generated responses. The overall pipeline is illustrated in Figure 4.2.



### Figure 4.2: The Upgraded Pipeline

Additionally, we optimize prompt construction by leveraging LangChain’s

`ChatPromptTemplate`<sup>4</sup> and `FewShotChatMessagePromptTemplate`<sup>5</sup>. By using these tools, the prompt can be organized into the system instructions and few shot examples (if applicable). In all system instruction prompts, we use `Pydantic` object format instructions to specify the desired output format, as defined in the variable `format_instructions`.

To formalize our definition, we are going to use several symbols within this section, where KG-specific objects will be denoted by the use of calligraphic font (such as  $\mathcal{E}$ ) and general objects will be denoted by the use of normal text font (such as  $E$ ).

#### 4.2.1 Translation

To enable multilingual support, a dedicated translation stage is introduced to convert user queries into English. This step is essential because the system is designed to operate on an English-based knowledge graph. The translation is handled using the `googletrans` library.

The process begins by determining the language of the users input. If the query is already in English, no translation is performed, and the original question is passed directly to the pipeline. However, if the query is in another language, it is translated into English before being processed by the system. This translated English version of the question then serves as the input for the subsequent stages of the pipeline, ensuring compatibility with the system's architecture while supporting multilingual capabilities.

The decision to use the `googletrans` library instead of relying on large language model (LLM) calls for translation was driven by practical considerations. LLM-based translations can be significantly more costly due to the computational resources required for each call. Additionally, using an LLM introduces a high dependency on the multilingual capabilities of the specific LLM being used, which may vary across different languages. By opting for `googletrans`, we ensure a lightweight, cost-effective solution that maintains sufficient translation quality for the system's requirements.

It is worth noting that while the `googletrans` library itself is open source, it uses the

---

<sup>4</sup>[https://python.langchain.com/api-reference/core/prompts/langchain\\_core.prompts.chat.ChatPromptTemplate.html](https://python.langchain.com/api-reference/core/prompts/langchain_core.prompts.chat.ChatPromptTemplate.html)

<sup>5</sup>[https://api.python.langchain.com/en/latest/prompts/langchain\\_core.prompts.few\\_shot.FewShotChatMessagePromptTemplate.html](https://api.python.langchain.com/en/latest/prompts/langchain_core.prompts.few_shot.FewShotChatMessagePromptTemplate.html)

Google Translate service/API, which is proprietary and not open source. If avoiding the use of proprietary services is a priority, an alternative is to use a fully open-source library such as `argostranslate`. However, the performance and accuracy of `argostranslate` are currently unknown, as it has not been tested in our system. Another option is to leverage LLMs for translation though this would come at the expense of higher computational costs as previously mentioned.

#### 4.2.2 Entities Linking

This stage is the combination of the Entities Extraction (Subsubsection 4.1.1), the Entity ID Retrieval (Subsubsection 4.1.2), as well as the Entity Disambiguation and Selection (Subsubsection 4.1.3) in the initial pipeline. We leverage generative LLM through in-context learning with few-shot prompting to extract a set of relevant entities  $E$  that appear in the question  $q$ . After that, we aim to find a set of top- $k$  most relevant entities  $\mathcal{E}_i$  from the KG based on every entity  $e_i \in E$ . Formally, we define  $E = \{e_1, e_2, \dots, e_n\}$  and  $\mathcal{E} = \bigcup_{i=1}^n \mathcal{E}_i = \bigcup_{i=1}^n \text{top}K(e_i)$ . Furthermore, since  $|E| = n$ ,  $|\mathcal{E}| \leq (n \times k)$  will hold, with  $|\mathcal{E}| = (n \times k)$  iff  $\bigcap_{i=1}^n \mathcal{E}_i = \emptyset$ .

The methods of finding the corresponding entities can be adjusted accordingly based on the specifications of the KG. For open and remotely-hosted KGs, we can leverage the provided APIs (if any) to find the entity candidates based on the given keyword, such as Wikidata API<sup>6</sup> (as implemented in Subsection 4.1.2) and DBpedia Lookup<sup>7</sup> for Wikidata and DBpedia, respectively. Subsequently, LLM is employed to obtain the most appropriate entity based on the question and the entity candidates using zero-shot prompting. On the other hand, for local KGs, semantic search can be employed by leveraging the text embeddings. Another suitable approach for small-scale KGs is to augment the prompt of the entity extraction process with the set of entities defined in the KG directly. However, we will only focus on the former approach in this work.

The system instruction prompt for the entities extraction and selection, as well as several few-shot examples, can be seen below. For the few-shot examples, we expand the set of few-shot examples defined in the initial pipeline to a total of 25 input-output pairs. We provide some examples in Table 4.1.

<sup>6</sup><https://www.wikidata.org/w/api.php>

<sup>7</sup><https://lookup.dbpedia.org/api/search>

### New Entities Extraction Prompt

- You are an entity extractor.
- Extract the entities from the given question! DO NOT hallucinate and only provide entities that are present in the question.
- These entities usage is to find the most appropriate resource URI from dbpedia to be used in SPARQL queries.
- If there is no entity in the question, return empty list.
- Sort the entities based on the importance of the entity in the question.
- ONLY return the entities. DO NOT return anything else.
- DO NOT include adjectives like 'Highest', 'Lowest', 'Biggest', etc in the entity.
- DO NOT provide any extra information, for instance explanation inside a brackets like '(population)', '(area)', '(place)', '(artist)', etc
- DO NOT include any explanations or apologies in your responses.
- Remove all stop words, including conjunctions like 'and' and prepositions like 'in' and 'on' from the extracted entity.
- Make the entity singular, not plural. For instance, if the entity is foods, then transform it into food.
- Even if there is only one entity, always return as a list.

Based on the query given, extract the entities from it and return the extracted entities in the format below.

{format\_instructions}

### New Entities Selection Prompt

For the entity given, find the most appropriate entity ID from the list of retrieved entities given to be used in Wikidata queries to answer the given question! ONLY return the entity ID from the list of retrieved entities given. DO NOT return anything else and DO NOT hallucinate. DO NOT include any explanations or apologies in your responses.

Based on the entity given, get the most appropriate entity ID from it and return the ID in the format below.

{format\_instructions}

Input	Output
Who wrote the Game of Thrones theme?	<code>{"names": ["Game of Thrones"]}</code>
How many movies did Park Chan-wook direct?	<code>{"names": ["Park Chan-wook"]}</code>
Give me all companies in the advertising industry.	<code>{"names": ["Company", "Advertising"]}</code>
How many films did Leonardo DiCaprio star in?	<code>{"names": ["Leonardo DiCaprio"]}</code>
In which films did Julia Roberts as well as Richard Gere play?	<code>{"names": ["Julia Roberts", "Richard Gere"]}</code>

**Table 4.1:** Several Few-Shot Examples of Entity Extraction

### 4.2.3 Retrieval

There are two types of questions, each requiring a distinct retrieval method. The first category consists of questions with a narrower focus, centered around a specific entity and relying on one-hop information. The second category includes questions with a broader scope and requires more complex query patterns, often involving aggregate functions such as `COUNT`. We refer to the former category as *simple questions*, and the latter as *complex questions*.

#### 4.2.3.1 Simple Question

Generating SPARQL query using LLM is not quite effective, especially on LLMs with a smaller amount of parameters like `Llama-3.1-8B`, despite it performs excellently on huge, commercial LLMs like `gpt-4o` (Emonet et al., 2024). It is not efficient as well, due to the usage of LLMs which requires a significant amount of time and computational resources for inference. Thus, in answering simple questions, we avoid using SPARQL query generation directly. Instead, we opt to answer the question through the text retrieval method first. Specifically, the first step is to extract the most relevant entity  $x$  from  $\mathcal{E}$  as obtained in the preceding stage using generative LLM. Next, a single-level (or known as single-hop) breadth-first traversal (BFS), will be employed to acquire the immediate neighbors of  $x$  in the conditions where  $x$  acts as an object (`?s ?p x`) and as a subject (`x`

?p ?o). The reason why we limit to the first depth is that most of the users' information needs can be resolved by one triple pattern only, which is evidenced by the fact that most of the top-100 frequently asked questions on Google are simple questions<sup>8</sup> (i.e., a question consists of a single triple pattern). Furthermore, these simple questions often serve as foundational elements in constructing more complex questions (Yani and Krisnadhi, 2021).

From this set of triples, we will perform *template-based verbalization* to convert each triple  $(s, p, o)$  into a sentence, with the following rule:  $\{s\}'s \{p\} \text{ is } \{o\}$ . To improve efficiency and avoid disambiguity, we ensure that the triples we aim to verbalize have unique properties in each case where  $x$  serves as both subject and object. For instance, consider the KG in Figure 2.16, where entity `:LionelMessi` has connections to three different entities with the same property `:hasChild`. Here, only one sentence will be constructed, based on the first matching triple, such as `Lionel Messi's has child is Thiago Messi`. Do note that even if the sentence has grammatical inaccuracies, the embeddings can still effectively capture the intended semantics.

The user's question and all verbalized triples are then encoded into embeddings to conduct a semantic search with QA objective, i.e., determining which generated sentence is the most probable answer to the user's question. The answer will be returned iff the similarity score exceeds a certain threshold  $\tau_2$ . Otherwise, the system proceeds with SPARQL query generation as in Subsubsection 4.2.3.2 (e.g., for questions with multiple triple patterns or incorporating aggregate functions but being misclassified). To standardize results for evaluation, the answer sentence is not returned immediately. Instead, we output the object (or subject) of the selected triple, simulating the result of a SPARQL query execution, which only comprises entities or literals.

To give a clear example, the illustration of template-based verbalization can be seen in Figure 4.3, which is equivalent to retrieval using SPARQL query as in Code 4.1.

---

<sup>8</sup><https://keywordtool.io/blog/most-asked-questions/>

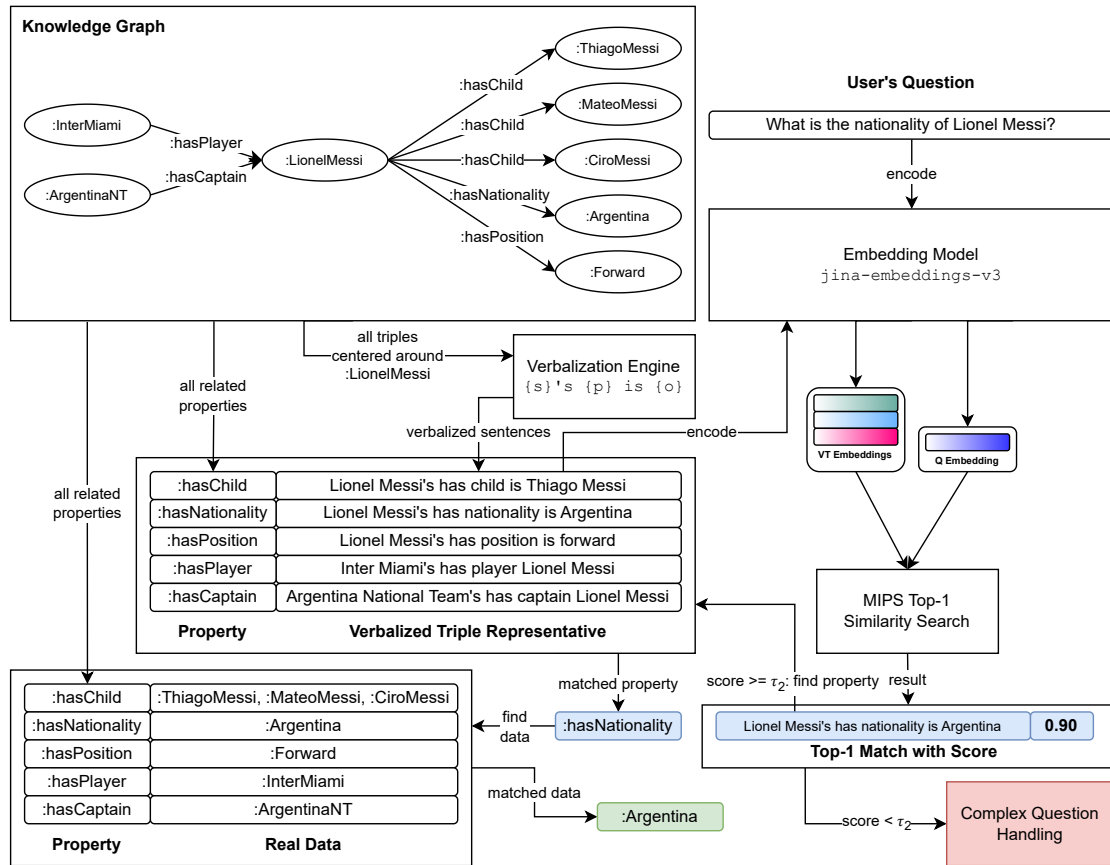


**Figure 4.3:** Template-Based Verbalization Overview

```

1 SELECT ?nationality WHERE {
2 :LionelMessi :hasNationality ?nationality .
3 }

```

**Code 4.1:** Equivalent SPARQL Query to Retrieval in Figure 4.3

#### 4.2.3.2 Complex Question

The aim of this stage is to generate an appropriate SPARQL query using generative LLM to answer the given question, similar to the SPARQL Query Generation stage (Subsubsection 4.1.4) in the initial pipeline. In order to do so, the resources needed to generate a query, i.e., entities, properties, or classes, have to be explicitly stated in the prompt. Previously, we have acquired  $\mathcal{E}$  as in Section 4.2.2. The next step is to retrieve the most relevant properties and classes that might appear in the query. A feasible way to perform this is by employing a hybrid search over sets of property labels  $\mathcal{P}$  and class

labels  $C$  as defined in the KG. However, we realize that comparing a long question like “What is the birth date of Tom Cruise?” with its appropriate short property label “birth date” will not produce an accurate result. Hence, we segment the question  $q$  into smaller chunks by leveraging n-gram, with the output  $N$  as the set of n-grams.

Unfortunately, the n-gram approach might not work accurately in the case where properties are not explicitly stated in the question. Thus, we employ another approach that involves generating property candidates  $P_G$  from  $q$  solely based on its question structure, regardless of how it is represented in the specific KG. With this approach, we hope to find the “starred in” property from the question “What films does Tom Cruise play in?”, as the property is not directly included in the question. This can be accomplished by using generative LLM through in-context learning with few-shot prompting. The system instruction prompt used for this can be seen below and the few-shot examples in Table 4.2.

#### Related Properties Generation Prompt (DBpedia)

Generate 3 DBpedia property that is appropriate to answer the given question by user. DO NOT include any explanations or apologies in your responses. No pre-amble.

Answer it in the format below.

{format\_instructions}

Input	Output
Who wrote the Game of Thrones theme?	<code>{"properties": ["composer", "music", "composition"]}</code>
How many movies did Park Chan-wook direct?	<code>{"names": ["Park Chan-wook"]}</code>
Give me all companies in the advertising industry.	<code>{"properties": ["company", "industry"]}</code>
How many films did Leonardo DiCaprio star in?	<code>{"properties": ["starring", "film", "actor"]}</code>
In which films did Julia Roberts as well as Richard Gere play?	<code>{"names": ["Julia Roberts", "Richard Gere"]}</code>

**Table 4.2:** Several Few-Shot Examples of Related Properties Generation (DBpedia)

We then define  $X$  as the set containing all property and class candidates retrieved in the preceding stage, i.e.,  $X = N \cup P_G$ . For each candidate  $x \in X$ , a hybrid search is conducted within each vector database  $V(\mathcal{P})$  and  $V(\mathcal{C})$ , which stores information about  $\mathcal{P}$  and  $\mathcal{C}$ , respectively. The candidates are selected iff their similarity scores exceed a certain threshold  $\tau_1$ . Auxiliary vector databases may be also constructed if needed, especially in the case where the property is separated into object property and data property, like in DBpedia's T-Box Ontology<sup>9</sup>.

The same as in the previous stage, if the KG is small enough, the property and class candidates might be included directly in the SPARQL generation prompt. We leverage in-context learning through few-shot and chain-of-thought prompting (generated by the GPT-4o model of ChatGPT) to enable the LLM to generate accurate SPARQL. The generated SPARQL query is designed to return a single-column result, containing either URIs or literals only. The system instruction prompt is provided below, while several few-shot examples for DBpedia are shown in Code 4.2. Do note that in the prompt, `resources` refers to a set of entities, whereas `ontology` refers to a set of properties and classes.

#### New SPARQL Generation Prompt (DBpedia)

```
INSTRUCTIONS

You are an assistant trained to generate DBpedia SPARQL queries. Use
the provided context to generate a valid SPARQL query. Based on the
given entities and properties context, please generate a valid SPARQL
query to answer the question! Use the URI from resources given if you
need to query more specific entity. On the other hand, USE classes from
ontology if it's more general. Return only the uri or literal only.
Always generate following the format instruction.

CONTEXT
- Entity: {resources}
- Ontology candidates:
{ontology}
```

<sup>9</sup><https://www.dbpedia.org/resources/ontology/>

```
FORMAT INSTRUCTIONS
{format_instructions}
```

```
1 [
2 {
3 "input": ""How many inhabitants does the largest city in Canada have?"" ,
4 "output": {
5 "thoughts": [
6 ""1. The question asks for the population of the largest city in
7 Canada."" ,
8 ""2. To retrieve this information from DBpedia, identify the correct URI
9 for 'Canada' and the relevant properties associated with cities and
10 population."" ,
11 ""3. From the context, 'Canada' has multiple possible URIs, with
12 'dbr:Canada' being the correct general URI."" ,
13 ""4. The term 'largest city' suggests we should find the entity linked
14 to Canada that represents its largest city, which requires identifying
15 the property that connects 'Canada' to its largest city."" ,
16 ""5. From the list of ontology candidates, 'dbo:largestCity' is the most
17 relevant object property for retrieving the largest city of a place,
18 with both the domain and range indicating populated places."" ,
19 ""6. After obtaining the largest city, we need to retrieve its population.
20 The 'dbo:populationTotal' data property provides the total population
21 for a city and is appropriate for this purpose."" ,
22 ""7. Formulate the query to first find the largest city of Canada using
23 'dbo:largestCity' and then retrieve its population using
24 'dbo:populationTotal'."" ,
25],
26 "sparql": ""SELECT DISTINCT ?num WHERE { dbr:Canada dbo:largestCity ?city .
27 ?city dbo:populationTotal ?num }"" ,
28 },
29 },
30 {
31 "input": ""What is the population of Cairo?"" ,
32 "output": {
33 "thoughts": [
34 ""1. The question seeks the population of 'Cairo'."" ,
35 ""2. From the context, there are multiple resources labeled 'Cairo', but
36 the correct URI for Cairo, Egypt, is 'dbr:Cairo'."" ,
37 ""3. To find the population of Cairo, identify a relevant data property that
38 retrieves population information."" ,
39 ""4. In the ontology candidates, 'dbo:populationTotal' is a suitable
40 data property to retrieve the total population of a populated place."" ,
41 ""5. Formulate the query to select the population of 'dbr:Cairo' using the
42 'dbo:populationTotal' property."" ,
43],
44 "sparql": ""SELECT ?pop WHERE { dbr:Cairo dbo:populationTotal ?pop }"" ,
45 },
46 },
47 {
```

```

48 "input": "How high is the Yokohama Marine Tower?",
49 "output": {
50 "thoughts": [
51 "1. The question is asking for the height of 'Yokohama Marine Tower'.",
52 "2. The context provides a single URI for 'Yokohama Marine Tower',
53 specifically 'dbr:Yokohama_Marine_Tower', which can be used directly
54 in the query.",
55 "3. To retrieve the height, identify a data property related to the
56 height or elevation of a structure.",
57 "4. In the ontology, 'dbo:height' is a relevant data property that does
58 not have a specific domain or range but matches the information
59 requested.",
60 "5. Formulate the query to select the height of 'dbr:Yokohama_Marine_Tower'
61 using the 'dbo:height' property.",
62],
63 "sparql": "SELECT DISTINCT ?num WHERE { dbr:Yokohama_Marine_Tower
64 dbo:height ?num }",
65 },
66 },
67]

```

**Code 4.2:** Several Few-Shot Examples of SPARQL Generation (DBpedia)

Lastly, the SPARQL query will be executed over the KG to obtain the relevant information, which result is also the output of this stage.

#### 4.2.4 Answer Generation

After retrieving relevant data from the previous step, the final step is to generate the answer in natural language, like in Subsubsection 4.1.5. However, the data retrieved in the preceding stage is still in the URI format. Thus, we have to resolve the label of each of those URIs prior to the answer generation stage. We obtain the label of each URI using the query in Code 4.3 for Wikidata and the query in Code 4.4 for DBpedia or other KGs that utilize `rdfs:label` to store information about labels of the resources.

```

1 SELECT ?item ?itemLabel WHERE {
2 # change these below to your desired resources
3 VALUES ?item { wd:Q31 wd:Q123 }
4 SERVICE wikibase:label {
5 bd:serviceParam wikibase:language "[AUTO_LANGUAGE],en" . }
6 }

```

**Code 4.3:** Wikidata SPARQL Query to Obtain Labels

```

1 SELECT ?item ?label WHERE {
2 # change these below your desired resources
3 VALUES ?item { dbr:Belgium dbr:Germany }
4 ?item rdfs:label ?label .
5 FILTER (lang(?label) = "en")
6 }

```

**Code 4.4:** DBpedia (or other KGs) SPARQL Query to Obtain Labels

We use generative LLM to generate the answer through in-context learning with the original user question and retrieved data provided as the context in the generation prompt. If the user's query is in a language other than English, we explicitly include the instruction "Answer in {user's language} language." within the prompt to direct the LLM to generate the response in the user's language. This ensures seamless multilingual support and enhances the user experience by delivering answers in their preferred language.

Below is an example of a prompt used for DBpedia, where `dbpedia_context` contains the retrieved information from the previous pipeline stage and the user's language is Indonesian.

#### Answer Generation Prompt (DBpedia)

You are an assistant for question-answering tasks. Use the following pieces of retrieved context to answer the question. Do not say "according to the context" or something like that, just answer directly with full sentence to the question using the context. If you don't know the answer, just say that you don't know. Use three sentences maximum and keep the answer concise. Answer in Indonesian language.

Question: {input}

Context: {dbpedia\_context}

Answer:

### 4.3 Libraries and Resources

In developing the system, we use Python programming language due to its rich ecosystem of libraries and frameworks that are suitable for building a RAG system. Apart from the default Python's libraries, we leverage these libraries in constructing both pipelines.

1. `pandas`: A powerful library for data manipulation and analysis. It provides data structures like `DataFrame` and `Series`, which make it easy to handle structured data and perform operations like filtering, grouping, and merging.
2. `transformers`: A popular library from HuggingFace that facilitates the loading, training, and fine-tuning of state-of-the-art large language models (LLMs). It supports numerous pre-trained models and can seamlessly integrate them into local systems for NLP tasks.
3. `torch`: PyTorch is an open-source deep learning framework that provides tools for building and training neural networks.
4. `weaviate`: An open-source vector search engine that supports hybrid search, allowing for both traditional keyword-based and semantic search using embeddings.
5. `nltk`: The Natural Language Toolkit is a comprehensive library for text processing. It provides tools for tokenizing text, generating n-grams, removing stopwords, and performing other NLP tasks.
6. `sentence-transformers`: A library for generating dense vector representations (embeddings) of phrases, sentences, paragraphs, or even documents.
7. `rdflib`: A library for working with RDF (Resource Description Framework) graphs. It allows for the creation, manipulation, and querying of RDF data, making it essential for handling knowledge graphs and linking data in a structured manner.
8. `langchain`: A framework designed to simplify the development of applications using LLMs. It provides tools for chaining multiple LLM operations together and creating complex prompt templates that can be reused across different tasks.
9. `sparqlwrapper`: A Python library for querying SPARQL endpoints to interact with RDF-based knowledge graphs. It allows for executing SPARQL queries on remote endpoints and retrieving data in various formats.
10. `pydantic`: A data validation library that enforces type checks and ensures that data is consistent and meets the defined schema.
11. `urllib`: A Python library for working with URLs. It provides tools for parsing, manipulating, and retrieving data from URLs.
12. `xmldict`: A library to parse and work with XML data by converting it into Python dictionaries. This simplifies the extraction of data from XML files, a common format for structured data exchange.
13. `requests`: A simple and elegant HTTP library for Python, making it easy to send

HTTP requests and handle responses. It supports various methods like GET, POST, PUT, DELETE, and is essential for interacting with web APIs.

14. `dotenv`: A library for loading environment variables from `‘.env’` files into the environment. This helps manage configuration settings securely and ensures sensitive information is not hardcoded in the code.
15. `numpy`: A library for numerical computing in Python. It provides support for multi-dimensional arrays, matrices, and mathematical functions.
16. `googletrans`: A library for translating text between multiple languages using the Google Translate API. It supports automatic language detection and facilitates seamless integration of multilingual capabilities into applications.

#### 4.4 Computational Setup

We conducted the evaluation using several GPU-supported instances rented on Vast.ai<sup>10</sup>. Specifically, we utilized two GPUs, the A100 and A800, each with 80GB of VRAM, depending on the availability. The hourly costs ranged from \$1 to \$1.60, resulting in a total development cost of \$100 (approximately IDR 1,661,326 including additional fees).

To streamline the setup process, we utilized the PyTorch (cuDNN Runtime) template available on Vast.ai. The steps to initiate an instance were as follows:

1. **Selecting a GPU:** Navigate to the “Search” page, select the desired GPU type (e.g., A100 or A800), and rent the instance.
2. **Launching the Instance:** Go to the “Instances” page and click “Open” on the instance just rented.

The rented instances come pre-configured with essential software such as Jupyter Notebook and PyTorch, enabling a quick start. After accessing the instance, we only needed to install additional libraries as specified in Subsection 4.3, ensuring a seamless environment for running our experiments.

---

<sup>10</sup><https://vast.ai/>

## CHAPTER 5

### RESULT AND ANALYSIS

In this chapter, we will discuss our experiment results and provide a thorough analysis.

#### 5.1 Experiment Results

We conduct several experiments both on Pipeline v1 (Subsection 4.1) and Pipeline v2 (Subsection 4.2) using different configurations. We did not evaluate the result of the RAG directly. Instead, we evaluated the intermediate result, i.e., the retrieved result of the KG querying using Jaccard Similarity, as explained in Subsection 3.6. Additionally, we evaluated the system holistically rather than focusing on individual components. The information provided in this subsection will be organized based on the used dataset.

##### 5.1.1 Dataset Pipeline v1

Using the dataset for the first pipeline, we conduct experiments on both pipelines with two LLMs: Mistral 7B and Mistral NeMo. For the Mistral 7B itself, we run the experiment on two settings: using HuggingFace’s Serverless Inference API<sup>1</sup> and running the LLM locally. The experiment result can be seen in Table 5.1.

Configuration	Jaccard Similarity
Pipeline v1 w/ Mistral 7B (API)	0.473
Pipeline v1 w/ Mistral 7B (Local)	0.554
Pipeline v1 w/ Mistral NeMo 12B (Local)	0.581
<b>Pipeline v2 w/ Mistral NeMo 12B (Local)</b>	<b>0.677</b>

**Table 5.1:** Experiment Results on the First Pipeline’s Dataset

The API version of Mistral 7B yields a different result than the locally-hosted model, despite both models using the same configuration. Since the implementation details of HuggingFace’s Serverless Inference API are undisclosed, our guess is that the API service might employ additional configurations that differ from the local setup. These could

---

<sup>1</sup><https://huggingface.co/docs/api-inference/index>

include runtime optimizations or difference in hardware environments, which may affect the final results. Additionally, we found that the API service is sometimes “unstable”; while it occasionally produces accurate tokens, at other times, it generates nonsensical outputs. We observed that posing the same question to the exact same system produced different results over time. For instance, when the system was being tested in August 2024, the system with Mistral 7B API usage produced a good result of 0.716. However, for the December 2024 evaluation, the performance of Mistral 7B API deteriorates to just 0.473, whereas the local setup achieves a result of 0.554. This performance discrepancy might be influenced by undisclosed configurations or runtime factors specific to the API service. The observed instability could also contribute to the lower performance, but further experiments are needed to confirm a causal link.

On the other hand, Mistral NeMo demonstrates a better performance compared to the original Mistral 7B, achieving a result of 0.581. We can infer that this improvement is likely attributable to factors such as its larger number of parameters (12B) compared to Mistral 7B (7B), which may enhance its capacity to learn complex relationships and generate more accurate predictions.

Lastly, testing the data with the same configuration on the second pipeline provides a justification that the second pipeline outperforms the first one, with a result of 0.677. One of the main contributors to this improvement is the inclusion of additional stages, such as verbalization-based retrieval (Subsubsection 4.2.3.1), in the second pipeline. This stage helps cover the limitation of LLMs in generating SPARQL queries, especially for simple questions, leading to a more robust pipeline overall. For instance, the LLM fails to generate the appropriate SPARQL query for the question “location name of the Mona Lisa”. The LLM is supposed to generate the SPARQL query as in Code 5.1, but it produces the SPARQL query as in Code 5.2. The latter query is entirely irrelevant because the property `wdt:P1810` refers to “subject named as”, which is unrelated to the intended property `wdt:P131`, meaning “located in the administrative territorial entity.” Meanwhile, by using verbalization, the property `wdt:P131` can be obtained correctly.

```

1 SELECT ?locationLabel WHERE {
2 wd:Q12418 wdt:P131 ?location .
3 SERVICE wikibase:label {
4 bd:serviceParam wikibase:language "[AUTO_LANGUAGE],mul,en".

```

```

5 }
6 }

```

**Code 5.1:** Ground Truth of “location name of the Mona Lisa”

```

1 SELECT ?itemLabel ?subjectName WHERE {
2 wd:Q12418 wdt:P1810 ?subjectName.
3 SERVICE wikibase:label {
4 bd:serviceParam wikibase:language "[AUTO_LANGUAGE],mul,en".
5 }
6 }

```

**Code 5.2:** Prediction of “location name of the Mona Lisa”

### 5.1.2 Dataset Pipeline v2

As demonstrated in Subsection 5.1.1, the second pipeline which utilizes Mistral NeMo outperforms the initial pipeline. Therefore, this pipeline is established as the baseline model for the next experiments. We conducted evaluations using four large language models (LLMs): Mistral NeMo, LLaMA 3.1 8B, Qwen2.5 Coder 7B, and Qwen2.5 7B. These experiments were performed on datasets tailored to specific knowledge base: the downsampled QALD-9-Plus dataset on Wikidata and DBPedia, and the Fasilkom UI’s Computer Science Major curriculum KG for local KG experiments. All the LLMs employed in this experiment are hosted locally.

#### 5.1.2.1 Wikidata

Configuration	Jaccard Similarity
Mistral NeMo 12B	0.423
LLaMA 3.1 8B	0.427
Qwen2.5 Coder 7B	0.428
<b>Qwen2.5 7B</b>	<b>0.458</b>

**Table 5.2:** Experiment Results on the Downsampled Wikidata QALD-9-Plus Dataset

The results for the Wikidata experiments are presented in Table 5.2. While Mistral NeMo, LLaMA 3.1 8B, and Qwen2.5 Coder 7B achieved comparable outcomes, Qwen2.5 7B demonstrated superior performance with a result of 0.458. This highlights Qwen2.5 7Bs ability to better align with the downsampled QALD-9-Plus dataset on Wikidata, outperforming the other models by a noticeable margin.

A closer analysis of the successes and failures highlights the strengths and limitations of the pipeline. For example, in the correct case of “In which films directed by Garry Marshall was Julia Roberts starring?”, the ground truth SPARQL query is `SELECT DISTINCT ?uri WHERE { ?uri wdt:P31 wd:Q11424 . ?uri wdt:P57 wd:Q315087 . ?uri wdt:P161 wd:Q40523 . }`. This query includes entities like `wd:Q11424` (film), `wd:Q315087` (Garry Marshall), and `wd:Q40523` (Julia Roberts). The pipeline successfully retrieved all relevant entities and properties, leading the LLM to generate the accurate SPARQL query `SELECT DISTINCT ?film WHERE { ?film wdt:P57 wd:Q315087 ; wdt:P161 wd:Q40523. }`. This success is attributed to effective entity extraction, property retrieval, and semantic understanding by Qwen2.5 7B.

Another notable instance of success pertains to questions involving aggregation functions, such as “How many programming languages are there?” The ground truth SPARQL query is `SELECT (COUNT(DISTINCT ?sub) AS ?count) WHERE { ?sub wdt:P31 wd:Q9143 . }`, which queries for the count of unique entities classified as instances of programming languages (`wd:Q9143`). The pipeline excelled by accurately retrieving the relevant entity (`wd:Q9143`, programming language). Since the property `wdt:P31` (instance of) is commonly used in Wikidata to indicate instance relationships, the system effectively leveraged its inherent knowledge to generate the correct SPARQL query with aggregation: `SELECT (COUNT(DISTINCT ?uri) AS ?count) WHERE { ?uri wdt:P31 wd:Q9143. }`. This example demonstrates the pipeline’s capability to process aggregate questions effectively, thanks to accurate entity-property pairing and semantic comprehension.

A further example showcasing the pipelines capabilities is the question, “In which programming language is GIMP written?” The ground truth SPARQL query is `SELECT DISTINCT ?uri WHERE { wd:Q8038 wdt:P277 ?uri . }`, where `wd:Q8038` refers

to GIMP, and `wdt:P277` denotes the “programming language” property. The pipeline succeeded due to its ability to verbalize the query in a meaningful way, generating the output: GIMP’s programmed in is C. This verbalization achieved a similarity score of 0.61, exceeding the required threshold of 0.6, and allowed the system to correctly identify the relevant entity, <http://www.wikidata.org/entity/Q15777>, which represents the C programming language.

However, the model also faced notable failures, revealing areas where the pipeline’s limitations impact performance. For example, in “Give me all Australian metalcore bands,” the ground truth query is `SELECT DISTINCT ?uri WHERE { ?uri wdt:P31 wd:Q215380 ; wdt:P495 wd:Q408 ; wdt:P136 wd:Q183862 . }`. This query requires entities like `wd:Q215380` (musical group), `wd:Q183862` (metalcore), and `wd:Q408` (Australia). Here, entity extraction only returned entities mentioned explicitly in the question—`['Australian', 'Metalcore Bands']`—which were then passed to the Wikidata API. Due to its keyword-based search mechanism, the API returned irrelevant entities like “Australian continent” and “Metalcore-bändide loen,” none of which matched the intended query. Additionally, the retrieved properties from the vector database failed to include `P495` (country of origin) because the question lacked explicit mentions of “country” or “origin,” highlighting the difficulty of capturing such semantic relationships. These limitations led to the generation of an incomplete or incorrect SPARQL query.

Another failure occurred with “Show a list of soccer clubs that play in the Bundesliga.” The ground truth SPARQL query is `SELECT DISTINCT ?uri WHERE { ?uri wdt:P31 wd:Q476028 ; wdt:P118 wd:Q82595 . }`. While the pipeline successfully retrieved all relevant entities (`wd:Q476028`, association football club; `wd:Q82595`, Bundesliga) and properties, the LLM mistakenly used `wdt:P6399` (Romanian Soccer Player ID) in the query, which is irrelevant to the question. The incorrectly generated SPARQL query was: `SELECT DISTINCT ?uri WHERE { ?uri wdt:P6399 ?id . ?uri wdt:P118 wd:Q82595 . }`. This error suggests that, while the vector database provided accurate properties, the LLM incorrectly prioritized a less relevant property, leading to an incorrect SPARQL query.

These examples highlight that the pipeline’s success relies heavily on accurate entity and property retrieval, as well as the LLM’s ability to prioritize the most relevant information.

A deeper analysis of these aspects will be provided in Subsection 5.2.4.

### 5.1.2.2 DBpedia

Configuration	Jaccard Similarity
Mistral NeMo 12B	0.450
LLaMA 3.1 8B	0.444
Qwen2.5 Coder 7B	0.442
<b>Qwen2.5 7B</b>	<b>0.517</b>

**Table 5.3:** Experiment Results on the Downsampled DBpedia QALD-9-Plus Dataset

The results for DBpedia are shown in Table 5.3. Similar to the Wikidata experiment, Qwen2.5 7B outperformed all other models with a result of 0.517. The higher performance of all models on DBpedia compared to Wikidata can likely be attributed to the inherent structure of DBpedia’s queries. Unlike Wikidata which relies heavily on ID-based URI naming (e.g., `wd:Q408` for “Australia” and `wdt:P36` for the property “capital”), DBpedia adopts a more human-readable, natural language-friendly URI naming (e.g., `dbr:Australia` for the entity and `dbo:capital` for the property). This more intuitive structure may contribute to the improved performance of the LLMs.

For instance, a correct example is the query, “Show a list of soccer clubs that play in the Bundesliga.”, which was previously misanswered in the Wikidata experiment. The ground truth SPARQL query is `SELECT DISTINCT ?uri WHERE { ?uri a dbo:SoccerClub ; dbo:league dbr:Bundesliga }`. In this case, the pipeline successfully retrieved all the necessary entities (`dbr:Bundesliga`), properties (`dbo:league`), and classes (`dbo:SoccerClub`) from both the API and the vector database. The LLM effectively utilized this information to generate the correct SPARQL query, highlighting the seamless integration of precise entity-property retrieval and the LLMs capability to construct semantically accurate outputs in the DBpedia context.

Another success case pertains to the question “How many programming languages are there?” with aggregation in DBpedia. The ground truth SPARQL query is `SELECT (COUNT(DISTINCT ?uri) AS ?c) WHERE { ?uri a dbo:ProgrammingLanguage }`. The pipeline successfully retrieved the entity `dbo:ProgrammingLanguage` from the

vector database and generated the correct SPARQL query: `SELECT (COUNT(DISTINCT ?uri) AS ?count) WHERE { ?uri a dbo:ProgrammingLanguage }`. This example showcases the pipeline’s ability to manage aggregate questions effectively by leveraging accurate entity-property retrieval and semantic comprehension to produce precise SPARQL queries.

Another success involved the question “In which programming language is GIMP written?”, where the ground truth SPARQL query is `SELECT DISTINCT ?uri WHERE { dbr:GIMP dbo:programmingLanguage ?uri }`. Here, the pipeline employed verbalization to generate the output: GIMP’s programming language is C (programming language). This verbalization achieved a similarity score of 0.74, surpassing the threshold of 0.6, and enabled the system to correctly identify the relevant entity, [http://dbpedia.org/resource/C\\_\(programming\\_language\)](http://dbpedia.org/resource/C_(programming_language)).

However, certain challenges persisted, mirroring limitations observed in the Wikidata experiments. For example, when answering the query “Give me all Australian metalcore bands,” the ground truth SPARQL query is `SELECT DISTINCT ?uri WHERE { ?uri a dbo:Band ; dbo:genre dbr:Metalcore { ?uri dbo:hometown dbr:Australia } UNION { ?uri dbo:hometown ?h . ?h dbo:country dbr:Australia } }`. While the pipeline successfully retrieved the entities `dbr:Australia` and `dbr:Metalcore`, it failed to retrieve the properties `dbo:hometown` and `dbo:country` because their semantic relationship to the question is not evident. This failure led to an incomplete SPARQL query, underscoring the challenge of capturing nuanced property associations in DBpedia.

Another failure occurred with the query “List all boardgames by GMT.” The ground truth SPARQL query is `SELECT ?uri WHERE { ?uri dbo:publisher dbr:GMT_Games }`. In this case, the entity extraction process identified “GMT,” which was then passed to the API. However, the API returned irrelevant entities such as “Greenwich Mean Time,” “UTC+07:00,” and “GMA Network,” instead of the correct entity, `dbr:GMT_Games`. This misalignment led the LLM to incorrectly interpret “GMT” as referring to “Greenwich Mean Time.” Although the pipeline successfully retrieved the relevant property (`dbo:publisher`), the inability to extract the correct entity ultimately caused the LLM to generate an inaccurate SPARQL query.

These examples reinforce that DBpedia’s natural-language-friendly structure can mitigate some challenges observed in Wikidata. However, the pipeline’s success still hinges on precise entity and property retrieval from the API and vector database.

5.1.2.3 Local KGs

Configuration	Jaccard Similarity
Mistral NeMo 12B	0.805
LLaMA 3.1 8B	0.778
Qwen2.5 Coder 7B	0.778
Qwen2.5 7B	0.805

Table 5.4: Experiment Results on the Local Curriculum KG Dataset

The results for the local knowledge graph (KG) experiments are shown in Table 5.4. Mistral NeMo and Qwen2.5 7B both achieved the highest performance with a score of 0.805. The remaining models, LLaMA 3.1 8B and Qwen2.5 Coder 7B, closely followed with scores of 0.778, differing by only one correctly answered question.

Notably, Qwen2.5 Coder 7B failed to answer the query, “What courses have Research Methodology and Scientific Writing as a prerequisite and a report document as an evaluation method, and are mandatory for faculty?”. Similarly, LLaMA 3.1 8B failed on the query, “How many prerequisite courses does ‘Algorithm Design and Analysis’ have?”. In contrast, Mistral NeMo and Qwen2.5 7B successfully handled both queries.

Closer analysis reveals that one recurring issue across all models was linked to the verbalization component. For example, in response to the question “How many evaluation methods of ‘Internet of Things’?”, the verbalized output was `Internet of Things’s has evaluation method is Task`. This result had a similarity score of 0.636, which exceeded the threshold of 0.6, leading the pipeline to incorrectly return `http://example.org/group_project` and `http://example.org/task` instead of aggregating them into the ground truth answer of 2. This highlights a limitation in the verbalization process, where overly simplistic or mismatched interpretations lead to errors.

Given these findings, it is still premature to declare a definitive best-performing model for the local KG dataset. To address this, all models will undergo further evaluation as part of the verbalization ablation study (see Subsubsection 5.2.1), enabling a deeper assessment of their capabilities and limitations.

#### 5.1.2.4 Multilingual Support

Configuration	Jaccard Similarity
Qwen2.5 7B	0.362

**Table 5.5:** Experiment Results on the Downsampled Wikidata QALD-9-Plus Dataset Indonesian Version

The multilingual support experiment focused exclusively on Qwen2.5 7B, the best-performing model in prior evaluations. This assessment utilized the downsampled Wikidata QALD-9-Plus dataset, which was translated into Indonesian to evaluate the model’s capability to process queries in a non-English language as mentioned in Subsubsection 3.5.2.1. As shown in Table 5.5, Qwen2.5 7B achieved a Jaccard similarity score of 0.362. While this performance was noticeably lower compared to the English version of the dataset (0.458), this result demonstrates that the model can process multilingual data, although with reduced accuracy.

In this experiment, Indonesian queries such as “Berapa banyak politisi yang lulus dari Universitas Columbia?” were first translated into English using the `googletrans` library, resulting in “How many politicians have graduated from Columbia University?” The system then processed this English query, retrieved relevant information from Wikidata, and generated the final answer directly in Indonesian, for example, “Ada 82 politisi yang telah lulus dari Universitas Columbia.” This workflow showcases the system’s ability to handle multilingual queries by combining translation and knowledge graph retrieval.

The performance gap between the English and Indonesian datasets can be attributed to several factors. A significant issue lies in the translation process itself. Queries translated from Indonesian back to English using the `googletrans` library were often paraphrased, introducing subtle but impactful discrepancies. For instance, the original English query is “Show me all the breweries in Australia.” and the Indonesian version is “Tunjukkan semua pabrik bir di Australia.” in the dataset. When translated back into English using

the library, it became “Show all beer factories in Australia.” This paraphrasing caused the system to fail in retrieving the correct entity (wd:Q131734, “brewery”), as the critical keyword “brewery” was missing in the back-translated query. These misalignments reduced the system’s ability to produce accurate results. Despite these challenges, the results indicate that the system possesses a foundational capability to handle multilingual queries, particularly in Indonesian.

### **5.1.3 General Analysis**

In this subsection, we will provide a general analysis of the experiment results from the perspectives of the LLMs and the knowledge bases.

#### **5.1.3.1 LLMs**

Based on the experiment results, Qwen2.5 7B emerges as the top-performing model across both Wikidata and DBpedia, achieving scores of 0.458 and 0.517, respectively. However, on the local KG dataset, this model performed on par with Mistral NeMo, and reached the highest score of 0.805. This consistent performance across datasets highlight Qwen2.5 7B’s versatility in both understanding natural language instructions and generating diverse types of outputs, including free text, structured data, and code.

This phenomenon is likely due to Qwen2.5 7B being pre-trained on up to 18 trillion tokens, compared to LLaMA 3.1 8B which was pre-trained on only 15 trillion tokens. The larger token count provides Qwen2.5 7B with broader exposure to diverse linguistic patterns, factual knowledge, and context, which enhances its capability in generating accurate responses. Additionally, Qwen Team (2024) reported that the 72B variant of instruction fine-tuned Qwen2.5 significantly outperforms the instruction fine-tuned LLaMA 3.1, which is in the similar 70B size range, on various benchmarking datasets. This indicates that Qwen2.5’s larger training data, combined with its optimization for instruction-following tasks, contributes to its superior performance over LLaMA 3.1 across multiple benchmarks.

On the other hand, while we initially thought that the Coder version of Qwen2.5 would outperform the general-purpose one, the evaluation results indicate the opposite. Qwen2.5 Coder 7B, which was trained on a large-scale, predominantly (70%) coding-related pre-training dataset comprising 5.2 trillion tokens, consistently produces

lower performance than the general-purpose Qwen2.5 7B. This could be attributed to the more specialized nature of the Coder version's training, which, while enhancing its expertise in code generation, might limit its broader understanding of general natural language tasks. In fact, the training of this Coder version comprised only 20% of general text data. However, we believe that tasks like generating SPARQL queries require a strong understanding of both the question and the given context, i.e., the associated properties and entities. This task demands not just coding proficiency but also a deeper grasp of natural language context, something the general-purpose model is better equipped to handle due to its exposure to a more diverse range of data.

In terms of Mistral NeMo, we are unable to draw definitive conclusions due to the lack of detailed information regarding its training data. However, it is plausible that the model's strong performance can be attributed to its significantly larger parameter count of 12B, compared to the other models, which only have 7B and 8B parameters. Larger models typically benefit from more capacity to capture complex patterns and relationships, which could contribute to their better performance. Without further information about its specific training dataset, it remains speculative, but the larger model size is a likely contributing factor to its success.

### 5.1.3.2 Knowledge Bases

In general, we found that there are two factors that affect the performance of the system from the perspective of the knowledge bases.

1. **URI Representation:** This refers to how resources are represented in the knowledge base using URIs. If the URIs are represented using natural language, like those in DBpedia and the local curriculum KG, we discovered that the model is likely to perform better. Since LLMs are trained to understand natural language, the more human-readable the URIs are, the easier the model to interpret and link entities to their corresponding properties. On the other hand, more cryptic URIs can hinder understanding and reduce performance.
2. **Size and Homogeneity:** The size and homogeneity of the KG can significantly impact model performance. Larger and more diverse KGs, like Wikidata and DBpedia, contain a greater number of entities, relationships, and concepts, which allows for a wider range of questions to be posed. While this expanded scope can enhance the

system’s versatility, it introduces significant challenges. As the variety of questions grows, the complexity of the corresponding SPARQL queries potentially increases, making them more difficult to generate. In contrast, smaller KGs, although offering less coverage than larger ones, present a more manageable scope. With a more limited range of possible questions, there is less need to generate complex and sophisticated SPARQL queries. As a result, this can lead to improved performance.

## **5.2 Ablation Study**

As outlined in Subsection 4.2, the second pipeline by default incorporates several key components: verbalization, chain of thoughts, few-shot examples, and entity and property retrieval from a vector database. In this section, we conduct an ablation study by systematically removing one component at a time to evaluate the significance of each component in the pipeline’s performance.

The experiments focus on the best-performing model for each knowledge base, which is Qwen2.5 7B. However, for the local KG dataset, as discussed in Subsubsection 5.1.2.3, all models are included in the verbalization evaluation to provide a comprehensive comparison.

### 5.2.1 Verbalization

Configuration	Knowledge Base	Jaccard Similarity
<b>Qwen2.5 7B</b>	<b>Wikidata</b>	<b>0.458</b>
Qwen2.5 7B w/o Verbalization	Wikidata	0.334
<b>Qwen2.5 7B</b>	<b>DBpedia</b>	<b>0.517</b>
Qwen2.5 7B w/o Verbalization	DBpedia	0.516
Mistral NeMo 12B	Curriculum KG	0.805
LLaMA 3.1 8B	Curriculum KG	0.778
Qwen2.5 Coder 7B	Curriculum KG	0.778
Qwen2.5 7B	Curriculum KG	0.805
<b>Mistral NeMo 12B w/o Verbalization</b>	<b>Curriculum KG</b>	<b>0.949</b>
LLaMA 3.1 8 w/o Verbalization	Curriculum KG	0.895
Qwen2.5 Coder 7B w/o Verbalization	Curriculum KG	0.922
<b>Qwen2.5 7B w/o Verbalization</b>	<b>Curriculum KG</b>	<b>0.949</b>

**Table 5.6:** Ablation Study Results on Verbalization

The results of the verbalization ablation study, shown in Table 5.6, reveal the critical role this component plays in generating accurate responses for SPARQL query tasks across various datasets. On Wikidata and DBpedia, the removal of verbalization led to notable declines in performance, with accuracy decreasing from 0.458 to 0.334 on Wikidata and from 0.517 to 0.516 on DBpedia. While the performance drop on DBpedia is marginal, these results underscore that for more complex and diverse KGs like Wikidata and DBpedia, verbalization plays a critical role in interpreting simpler local natural language questions. Sometimes, LLMs can get confused and fail to generate the appropriate single-hop SPARQL query, like the Mona Lisa example mentioned in Subsubsection 5.1.1. Verbalization serves as a crucial step in bridging the user's input with the model's knowledge of the graph structure by converting the extracted triples into human-readable sentences. This transformation enables the semantic alignment necessary for accurate query resolution.

In contrast, the results of the verbalization ablation study for the local curriculum KG dataset highlight the significant performance improvements achieved by removing

the verbalization component. As mentioned in Subsubsection 5.1.2.3, all models demonstrated growth in accuracy when verbalization was omitted. Mistral NeMo and Qwen2.5 7B achieved the highest scores at 0.949, while Qwen2.5 Coder 7B and LLaMA 3.1 8B followed closely with scores of 0.922 and 0.895, respectively.

The improvements underscore that the verbalization component acted as a bottleneck in the second pipeline for the curriculum KG dataset. While the verbalization step successfully retrieved relevant entities or properties, it lacked the ability to perform aggregate operations such as `COUNT`. This limitation resulted in errors for aggregation-based queries, as discussed in Subsubsection 5.1.2.3. Additionally, verbalization introduced occasional inaccuracies when it failed to retrieve suitable template sentences, further complicating its effectiveness for domain-specific tasks.

The results also demonstrate that for datasets like the curriculum KG, which rely on small ontology structures and aggregate operations, the SPARQL generation component aligns better with the dataset's characteristics. Since the questions in such datasets are generally straightforward and simple, it becomes easier to generate SPARQL queries directly. By bypassing verbalization, the models were able to directly utilize SPARQL queries, eliminating ambiguity and improving accuracy.

These findings suggest that for similar domain-specific KGs, careful evaluation of the verbalization component is essential. Verbalization may need to be omitted if it cannot effectively handle aggregation and retrieve appropriate templates; otherwise, it introduces unnecessary complexity. This process should only be employed if the question is single-hop and does not contain any aggregations.

### 5.2.2 Chain of Thoughts (CoT)

Configuration	Knowledge Base	Jaccard Similarity
<b>Qwen2.5 7B</b>	<b>Wikidata</b>	<b>0.458</b>
Qwen2.5 7B w/o Chain of Thoughts	Wikidata	0.436
<b>Qwen2.5 7B</b>	<b>DBpedia</b>	<b>0.517</b>
Qwen2.5 7B w/o Chain of Thoughts	DBpedia	0.455
Qwen2.5 7B	Curriculum KG	0.805
Qwen2.5 7B w/o Verbalization	Curriculum KG	0.949
Qwen2.5 7B w/o Chain of Thoughts	Curriculum KG	0.808
<b>Qwen2.5 7B w/o Verbalization &amp; Chain of Thoughts</b>	<b>Curriculum KG</b>	<b>0.976</b>

**Table 5.7:** Ablation Study Results on CoT

The ablation study presented in Table 5.7 illustrates the varying impact of removing the CoT component across different datasets. For Wikidata and DBpedia, removing CoT caused a performance drop, from 0.458 to 0.436 and from 0.517 to 0.455, respectively. This demonstrates that CoT plays a supportive role in improving the models ability to generate SPARQL queries for these datasets. The performance decline, although noticeable, is not drastic, suggesting that CoT is helpful but not strictly necessary. It can be viewed as an enhancement that bolsters query reasoning in more complex or open-ended scenarios, particularly for datasets requiring intermediate reasoning steps to construct accurate queries.

In contrast, the local curriculum KG dataset showed a different trend. Here, removing CoT had little to no negative impact, with performance slightly improving from 0.805 to 0.808 in the default configuration and from 0.949 to 0.976 when verbalization was also omitted. These results suggest that CoT may not be necessary for datasets with simpler or more direct queries, such as the curriculum KG. In such cases, the absence of CoT allows the model to focus directly on generating SPARQL queries without intermediate reasoning steps that might introduce ambiguity or unnecessary complexity. The simplicity of the questions in the curriculum KG, coupled with its smaller ontology, likely reduces the utility of CoT, making it redundant.

The study highlights an important insight: the utility of CoT depends on the dataset's characteristics. For datasets like Wikidata and DBpedia, which often involve more complex queries requiring intermediate reasoning, CoT can serve as a valuable tool to structure the model's thought process, leading to more accurate query generation. However, for smaller and simpler datasets like the curriculum KG, where queries are straightforward and do not require elaborate reasoning, CoT may hinder performance by adding unnecessary overhead.

These findings suggest that the inclusion of CoT should be carefully evaluated based on the dataset's complexity. For datasets with complex question structures, CoT can enhance query accuracy by guiding the model through a step-by-step reasoning process. On the other hand, for simpler datasets, omitting CoT can streamline the pipeline, reduce potential confusion, and improve efficiency.

### 5.2.3 Few Shots

Configuration	Knowledge Base	Jaccard Similarity
<b>Qwen2.5 7B</b>	<b>Wikidata</b>	<b>0.458</b>
Qwen2.5 7B w/o Few Shots	Wikidata	0.342
<b>Qwen2.5 7B</b>	<b>DBpedia</b>	<b>0.517</b>
Qwen2.5 7B w/o Few Shots	DBpedia	0.410
Qwen2.5 7B	Curriculum KG	0.805
<b>Qwen2.5 7B w/o Verbalization</b>	<b>Curriculum KG</b>	<b>0.949</b>
Qwen2.5 7B w/o Few Shots	Curriculum KG	0.724
Qwen2.5 7B w/o Verbalization & Few Shots	Curriculum KG	0.651

**Table 5.8:** Ablation Study Results on Few Shots

The results of the ablation study, shown in Table 5.8, underscore the critical role of the few-shots component in generating accurate SPARQL queries across datasets. Across Wikidata, DBpedia, and local curriculum KG, the removal of few shots led to consistent performance drops. For Wikidata, performance declined from 0.458 to 0.342, and for DBpedia, it dropped from 0.517 to 0.410. Similarly, the Curriculum KG dataset exhibited a substantial decline, with performance decreasing from 0.805 to 0.724 when few shots

were removed and from 0.949 to 0.651 when both verbalization and few shots were omitted. These consistent reductions highlight the utility of few shots as a crucial component for improving model performance across diverse datasets.

Few shots play a pivotal role in providing tailored examples that guide the LLM in generating SPARQL queries with the appropriate structure and format for each dataset. They help the model adapt to the specific requirements of the dataset, such as returning only URIs in QALD-9-Plus datasets or handling entity relationships and labels in domain-specific datasets like the curriculum KG. By offering examples that clarify these nuances, few shots enhance the models ability to align its outputs with the expectations of the dataset, ensuring greater accuracy and relevance.

The pronounced impact on the curriculum KG, a custom domain-specific dataset, further emphasizes the importance of few shots for datasets that the LLM lacks prior exposure to. Unlike open datasets with relatively standardized query formats, the local curriculum KG introduces unique structures and conventions that few shots help to elucidate. For instance, few-shot examples can demonstrate how to generate SPARQL queries that account for localized relationships or custom entity-label mappings, which are not inherent in the LLMs pre-trained knowledge.

Across all datasets, the inclusion of few shots bridges the gap between the general capabilities of the LLM and the specific needs of the task. While the performance improvements for open datasets like Wikidata and DBpedia may appear less dramatic compared to the local curriculum KG, the utility of few shots remains evident, serving as a mechanism to fine-tune the models query generation process. For custom datasets, they are indispensable, ensuring the model generates meaningful results tailored to the dataset's unique ontology and query requirements.

### 5.2.4 Entity and Property (Ontology) Retrieval from Vector Database

Configuration	Knowledge Base	Jaccard Similarity
<b>Qwen2.5 7B</b>	<b>Wikidata</b>	<b>0.458</b>
Qwen2.5 7B w/o Vector Database Retrieval	Wikidata	0.377
<b>Qwen2.5 7B</b>	<b>DBpedia</b>	<b>0.517</b>
Qwen2.5 7B w/o Vector Database Retrieval	DBpedia	0.381
Qwen2.5 7B	Curriculum KG	0.805
<b>Qwen2.5 7B w/o Verbalization</b>	<b>Curriculum KG</b>	<b>0.949</b>
Qwen2.5 7B w/o Vector Database Retrieval	Curriculum KG	0.183
Qwen2.5 7B w/o Verbalization & Vector Database Retrieval	Curriculum KG	0.000

**Table 5.9:** Ablation Study Results on Ontology Retrieval

The ablation study results in Table 5.9 emphasize the crucial role of the entity and property retrieval component from the vector database in enabling accurate SPARQL query generation across datasets. When this component is removed, the performance of the model drops significantly for all datasets. Specifically, the results decline from 0.458 to 0.377 on Wikidata, from 0.517 to 0.381 on DBpedia, and from 0.805 to a strikingly low 0.183 on the local curriculum KG dataset. Furthermore, in the curriculum KG, when both the verbalization and vector database retrieval components are omitted, the performance plummets to 0.000, demonstrating the model's complete inability to generate meaningful SPARQL queries in such scenarios.

The vector database retrieval component is crucial because it identifies the most relevant entities and properties needed to construct SPARQL queries based on the input question and context. Without this step, the model struggles to locate the correct entities and relationships, leading to inaccurate or incomplete queries. This issue is especially pronounced in the curriculum KG dataset, where the vector database serves as the primary source of knowledge about the dataset's unique structure and content. Since the LLM lacks prior training on the curriculum KG, omitting this component renders the model entirely ineffective, as evidenced by the 0.000 score.

While the performance drops on Wikidata and DBpedia are significant, they are not

as dramatic as in the curriculum KG. This disparity arises because open datasets like Wikidata and DBpedia are more likely to have been partially incorporated into the LLM's training data. Consequently, the model can leverage its preexisting knowledge of these datasets to some extent, even in the absence of vector database retrieval. However, the drop from 0.458 to 0.377 on Wikidata and from 0.517 to 0.381 on DBpedia still underscores the importance of this component in refining and contextualizing the queries for improved accuracy.

Interestingly, on the curriculum KG dataset, the model achieves a score of 0.183 when the vector database retrieval is removed but the verbalization component remains active. This suggests that the verbalization component can partially compensate for the absence of vector retrieval by utilizing natural language to infer some entity relationships. However, this approach is far less effective, as verbalization alone cannot handle complex queries involving precise entity and property relationships.

The complete failure of the model (0.000 score) when both vector database retrieval and verbalization are omitted from the curriculum KG highlights the critical interdependence of these components. Without vector retrieval, the model lacks foundational knowledge of the dataset, and without verbalization, it cannot even attempt to compensate through natural language reasoning. This outcome illustrates the importance of vector database retrieval as a key for working with domain-specific datasets that require tailored knowledge and entity alignment.

### **5.2.5 Conclusion**

Based on the aforementioned explanation, it can be concluded that the entity and property retrieval from the vector database has the most significant impact on the system's accuracy. Across all datasets, removing this component resulted in performance declines, particularly for the local curriculum KG dataset, where the accuracy plummeted to zero without this step. This highlights the importance of identifying relevant entities and properties to construct accurate queries, especially for local, domain-specific KGs where the LLM lacks prior exposure. Even for open KGs like DBpedia and Wikidata—data the LLM might have encountered during training—performance degradation remains unavoidable without context retrieval, highlighting its indispensable role.

### 5.3 Dataset Issues

Upon concluding the experiments and obtaining all the results, we noticed that there were some issues in the dataset, particularly the QALD-9-Plus Dataset.

1. **Inaccurate Ground Truth (1):** We observed that for the question “How many languages are spoken in Colombia?”, the ground truth provided an inaccurate DBpedia SPARQL query. Specifically, the ground truth query is `SELECT (COUNT(DISTINCT ?uri) AS ?c) WHERE { ?uri rdf:type dbo:Language . dbr:Colombia dbo:language ?uri }`. However, there was no direct connection between `dbr:Colombia` and `dbo:language`. As a result, the execution of the query returned 0, which did not make sense for such a question. On the other hand, our system successfully generated the correct query, `SELECT COUNT(DISTINCT ?lang) WHERE { ?lang dbo:spokenIn dbr:Colombia }`, which returned 97, the correct answer. Yet, since the supposed true answer did not match the ground truth, it was classified as incorrect.
2. **Inaccurate Ground Truth (2):** We observed that for the question “How many films did Hal Roach produce?” the ground truth query is `SELECT (COUNT(?uri) as ?c) WHERE { ?uri wdt:P162 wd:Q72792 . }`, which lacked specificity. While the question explicitly asks about films, the ground truth query did not restrict the query to instances of films. In contrast, our system generated the query `SELECT (COUNT(DISTINCT ?uri) AS ?count) WHERE { ?uri wdt:P31 wd:Q11424 ; wdt:P162 wd:Q72792 . }`, which correctly filtered for entities that are instances of films (`wd:Q11424`). However, due to the lack of specificity in the ground truth query, our accurate result was incorrectly classified as a mismatch.
3. **Ambiguous Question (1):** We observed that for the question “how big is the total area of North Rhine-Westphalia?”, it was ambiguous what unit the answer was expected to be in. The ground truth query was completely correct by presenting the result in squared metres using the query `SELECT ?tarea WHERE { dbr:NorthRhine-Westphalia dbo:areaTotal ?tarea }`. However, this became ambiguous since there were more than one properties related to total area connected to `dbr:NorthRhine-Westphalia`. Unfortunately, our verbalization system retrieved the other property, `dbo:PopulatedPlace/areaTotal` which presented the result in squared kilometres. As a result, our result was classified as incorrect, despite the

difference was only in unit.

4. **Ambiguous Question (2):** We also observed that the question “Give me all soccer clubs in Spain.” was ambiguous. Our system generated the query `SELECT DISTINCT ?uri WHERE { ?uri a dbo:SoccerClub ; dbo:country dbr:Spain }`, which was semantically correct. However, the ground truth is `SELECT DISTINCT ?uri WHERE { ?uri a dbo:SoccerClub { ?uri dbo:ground dbr:Spain } UNION { ?uri <http://dbpedia.org/property/ground> ?ground FILTER regex(?ground, "Spain") } }`, which seemed to be overly specific. As a result, our result was classified as incorrect.



## **CHAPTER 6**

### **CONCLUSION**

This chapter will discuss the summary of this research in building a GraphRAG framework consisting of only open-source components. Additionally, several potential future works are also addressed to improve this research in the future.

#### **6.1 Conclusion**

In this research, we implement a system for performing RAG over KGs using only open-source components. We design and develop two pipelines. The first pipeline is simpler but has a narrower scope, focusing exclusively on Wikidata. It consists of several stages: entity extraction and linking, SPARQL query generation (using the top-100 most frequently used properties augmented into the prompt), and answer generation. However, this approach limits the types of questions that the system can handle, since the properties are only obtained from the pre-defined top-100 properties list. To overcome this limitation, we introduce a second pipeline that builds upon the first. It incorporates verbalization-based retrieval to handle single-hop questions and integrates a vector database for flexible retrieval of ontological elements (properties and classes) without requiring explicit mention of all elements in the prompt. This enhancement significantly improves the system's ability to support a broader range of question types.

The final experiment demonstrates that the usage of Qwen2.5 7B LLM in the second pipeline serves as the best configuration for all datasets: Wikidata, DBpedia, and the local curriculum KG. This configuration yields accuracies of 0.458, 0.517, and 0.805 for Wikidata, DBpedia, and the curriculum KG, respectively. Our findings indicate that simpler KGs (and ontology) tend to yield better performance, as observed with the curriculum KG. Moreover, the multilingual support experiment demonstrated that the system could process queries in Indonesian, achieving an accuracy of 0.362. While this score is lower than that of the English dataset, it highlights the system's foundational ability to handle multilingual queries through translation and retrieval mechanisms.

Although the results for Wikidata and DBpedia appear relatively low, we believe their potential accuracy could be higher. This conclusion stems from our observation of

several inaccuracies within these datasets, which likely impacted the evaluation outcomes. Furthermore, we found that the ablation study underscores the critical role of entity and property retrieval in providing context to the LLM, significantly impacting the system's overall performance. Removing this component caused major accuracy drops across all datasets, with the curriculum KG dataset being the most affected, where accuracy fell to zero. Another interesting finding is that for simpler KGs, the use of verbalization deteriorates pipeline performance. This decline is attributed to the misinterpretation of questions and limitations in handling aggregate and other complex operations.

Finally, the objectives from Subsection 1.3 are completed throughout this research by:

1. Developing a RAG system using KG as the primary knowledge base, which pipelines are explained in Subsection 3.2 and Chapter 4.
2. Determining that Qwen2.5 7B is the best-performing LLM, as explained in Subsubsubsection 5.1.3.1.
3. Identifying that the most influential component within the architecture is the entity and property retrieval, which information is obtained through the ablation study (Subsection 5.2).

## 6.2 Future Works

Given the limitations of the current research, there are several possible future works.

1. Fine-tuning the LLM for SPARQL query generation task. This would allow the model to better understand the nuances of query structure, entity relationships, and contextual mapping from natural language to SPARQL, ultimately improving accuracy and performance in generating precise queries.
2. Develop pipeline native to specific language, such as Bahasa Indonesia. This would tackle current limitations in the translation component, which may not accurately capture the nuances of the local language. By developing a pipeline that natively supports Bahasa Indonesia, the system could more effectively process queries in the local language, enhancing performance and accuracy.
3. Exploring the use of LLMs with a larger number of parameters. Models with more parameters typically have enhanced capacity to understand and generate complex queries, potentially leading to improved performance in SPARQL query

generation. However, this approach would also require addressing challenges related to computational efficiency and resource requirements.

4. Introducing a dedicated classifier to determine whether a question can be resolved using verbalization (can be in the form of an intent classification task). This classifier would analyze the input question and decide whether verbalization is an appropriate approach to answer the question. By integrating this component, the system could optimize its query generation process, ensuring that verbalization is only applied when suitable, while alternative strategies are employed for more complex questions.
5. Scaling the system to larger knowledge graphs. The current implementation is limited by memory constraints due to the use of `rdflib` for in-memory graph storage. Future work could involve optimizing the system to handle larger knowledge graphs by implementing a persistent storage solution, such as a database-backed storage approach (e.g., using SQLite, GraphDB, or Virtuoso), to efficiently manage larger datasets and improve scalability.



## BIBLIOGRAPHY

- Abu-Rasheed, H., Weber, C., Zenkert, J., Dornhfer, M., and Fathi, M. (2022). Transferrable framework based on knowledge graphs for generating explainable results in domain-specific, intelligent information retrieval. *Informatics*, 9(1).
- Abujabal, A., Yahya, M., Riedewald, M., and Weikum, G. (2017). Automated template generation for question answering over knowledge graphs. In *Proceedings of the 26th International Conference on World Wide Web, WWW '17*, pages 1191–1200, Republic and Canton of Geneva, CHE. International World Wide Web Conferences Steering Committee.
- Ainslie, J., Lee-Thorp, J., de Jong, M., Zemlyanskiy, Y., Lebron, F., and Sanghai, S. (2023). GQA: Training generalized multi-query transformer models from multi-head checkpoints. In Bouamor, H., Pino, J., and Bali, K., editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 4895–4901, Singapore. Association for Computational Linguistics.
- Alammar, J. (2019). The illustrated gpt-2 (visualizing transformer language models). Jay Alammar Visualizing machine learning one concept at a time.
- Allemang, D. and Hendler, J. (2011). *Semantic Web for the Working Ontologist: Effective Modeling in RDFS and OWL*. Elsevier Science & Technology.
- Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., and Ives, Z. (2007). Dbpedia: A nucleus for a web of open data. In Aberer, K., Choi, K.-S., Noy, N., Allemang, D., Lee, K.-I., Nixon, L., Golbeck, J., Mika, P., Maynard, D., Mizoguchi, R., Schreiber, G., and Cudré-Mauroux, P., editors, *The Semantic Web*, pages 722–735, Berlin, Heidelberg. Springer Berlin Heidelberg.
- Ba, J. L., Kiros, J. R., and Hinton, G. E. (2016). Layer normalization.
- Bahdanau, D., Cho, K., and Bengio, Y. (2016). Neural machine translation by jointly learning to align and translate.
- Bast, H. and Haussmann, E. (2015). More accurate question answering on freebase. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management, CIKM '15*, pages 1431–1440, New York, NY, USA. Association for Computing Machinery.
- Bavarian, M., Jun, H., Tezak, N., Schulman, J., McLeavey, C., Tworek, J., and Chen, M. (2022). Efficient training of language models to fill in the middle.

- Beltagy, I., Peters, M. E., and Cohan, A. (2020). Longformer: The long-document transformer.
- Berant, J., Chou, A. K., Frostig, R., and Liang, P. (2013). Semantic parsing on freebase from question-answer pairs. In *Conference on Empirical Methods in Natural Language Processing*.
- Berners-Lee, T., Hendler, J., and Lassila, O. (2001). The semantic web: A new form of web content that is meaningful to computers will unleash a revolution of new possibilities. *ScientificAmerican.com*.
- Bizer, C., Lehmann, J., Kobilarov, G., Auer, S., Becker, C., Cyganiak, R., and Hellmann, S. (2009). Dbpedia - a crystallization point for the web of data. *Journal of Web Semantics*, 7(3):154–165. The Web of Data.
- Boley, H. and Kifer, M. (2010). A guide to the basic logic dialect for rule interchange on the web. *IEEE Transactions on Knowledge and Data Engineering*, 22(11):1593–1608.
- Both, A. and Perevalov, A. (2024). Towards llm-driven natural language generation based on sparql queries and rdf knowledge graphs. In *TEXT2KG/DQMLKG@ESWC*.
- Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. (2020). Language models are few-shot learners. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.
- Cambridge Semantics (2021). Sparql queries basics. Accessed: 2024-07-16.
- Cantallos, M. M., Sánchez-Alonso, S., and García-Barriocanal, E. (2019). A systematic literature review on wikidata. *Data Technol. Appl.*, 53:250–268.
- Chen, B., Chen, K., Yang, Y., Amini, A., Saxena, B., Chavez-Garcia, C., Babaei, M., Feizpour, A., and Varr, D. (2023). Towards improving the explainability of text-based information retrieval with knowledge graphs. Copyright - 2023. This work is published under <http://creativecommons.org/licenses/by/4.0/> (the License). Notwithstanding the ProQuest Terms and Conditions, you may use this content in accordance with the terms of the License; Last updated - 2023-01-19.

- Child, R., Gray, S., Radford, A., and Sutskever, I. (2019). Generating long sequences with sparse transformers.
- Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation.
- Choi, H., Kim, J., Joe, S., and Gwon, Y. (2021). Evaluation of bert and albert sentence embedding performance on downstream nlp tasks. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 5482–5487.
- Cur, O. and Blin, G. (2015). Chapter three - rdf and the semantic web stack. In *RDF Database Systems*, pages 41–80. Morgan Kaufmann, Boston.
- Dai, S., Zhou, Y., Pang, L., Liu, W., Hu, X., Liu, Y., Zhang, X., Wang, G., and Xu, J. (2024). Neural retrievers are biased towards llm-generated content. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '24, page 526537, New York, NY, USA. Association for Computing Machinery.
- Dao, T. (2023). Flashattention-2: Faster attention with better parallelism and work partitioning.
- Decker, S., Melnik, S., van Harmelen, F., Fensel, D., Klein, M., Broekstra, J., Erdmann, M., and Horrocks, I. (2000). The semantic web: the roles of xml and rdf. *IEEE Internet Computing*, 4(5):63–73.
- Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding.
- DeWeese, K. P. and Segal, D. (2014). *Libraries and the Semantic Web*. Springer International Publishing AG.
- Diefenbach, D., Lopez, V., Singh, K., and Maret, P. (2018). Core techniques of question answering systems over knowledge bases: a survey. *Knowledge and Information Systems*, 55(3):529–569.
- Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., Letman, A., Mathur, A., Schelten, A., Yang, A., Fan, A., Goyal, A., Hartshorn, A., Yang, A., Mitra, A., Sravankumar, A., Korenev, A., Hinsvark, A., Rao, A., Zhang, A., Rodriguez, A., Gregerson, A., Spataru, A., Roziere, B., Biron, B., Tang, B., Chern, B., Caucheteux, C., Nayak, C., Bi, C., Marra, C., McConnell, C., Keller, C., Touret, C., Wu, C., Wong, C., Ferrer, C. C., Nikolaidis, C., Allonsius, D., Song, D., Pintz, D.,

Livshits, D., Esiobu, D., Choudhary, D., Mahajan, D., Garcia-Olano, D., Perino, D., Hupkes, D., Lakomkin, E., AlBadawy, E., Lobanova, E., Dinan, E., Smith, E. M., Radenovic, F., Zhang, F., Synnaeve, G., Lee, G., Anderson, G. L., Nail, G., Mialon, G., Pang, G., Cucurell, G., Nguyen, H., Korevaar, H., Xu, H., Touvron, H., Zarov, I., Ibarra, I. A., Kloumann, I., Misra, I., Evtimov, I., Copet, J., Lee, J., Geffert, J., Vranes, J., Park, J., Mahadeokar, J., Shah, J., van der Linde, J., Billock, J., Hong, J., Lee, J., Fu, J., Chi, J., Huang, J., Liu, J., Wang, J., Yu, J., Bitton, J., Spisak, J., Park, J., Rocca, J., Johnstun, J., Saxe, J., Jia, J., Alwala, K. V., Upasani, K., Plawiak, K., Li, K., Heafield, K., Stone, K., El-Arini, K., Iyer, K., Malik, K., Chiu, K., Bhalla, K., Rantala-Yearly, L., van der Maaten, L., Chen, L., Tan, L., Jenkins, L., Martin, L., Madaan, L., Malo, L., Blecher, L., Landzaat, L., de Oliveira, L., Muzzi, M., Pasupuleti, M., Singh, M., Paluri, M., Kardas, M., Oldham, M., Rita, M., Pavlova, M., Kambadur, M., Lewis, M., Si, M., Singh, M. K., Hassan, M., Goyal, N., Torabi, N., Bashlykov, N., Bogoychev, N., Chatterji, N., Duchenne, O., elebi, O., Alrassy, P., Zhang, P., Li, P., Vasic, P., Weng, P., Bhargava, P., Dubal, P., Krishnan, P., Koura, P. S., Xu, P., He, Q., Dong, Q., Srinivasan, R., Ganapathy, R., Calderer, R., Cabral, R. S., Stojnic, R., Raileanu, R., Girdhar, R., Patel, R., Sauvestre, R., Polidoro, R., Sumbaly, R., Taylor, R., Silva, R., Hou, R., Wang, R., Hosseini, S., Chennabasappa, S., Singh, S., Bell, S., Kim, S. S., Edunov, S., Nie, S., Narang, S., Raparthy, S., Shen, S., Wan, S., Bhosale, S., Zhang, S., Vandenhende, S., Batra, S., Whitman, S., Sootla, S., Collot, S., Gururangan, S., Borodinsky, S., Herman, T., Fowler, T., Sheasha, T., Georgiou, T., Scialom, T., Speckbacher, T., Mihaylov, T., Xiao, T., Karn, U., Goswami, V., Gupta, V., Ramanathan, V., Kerkez, V., Gonguet, V., Do, V., Vogeti, V., Petrovic, V., Chu, W., Xiong, W., Fu, W., Meers, W., Martinet, X., Wang, X., Tan, X. E., Xie, X., Jia, X., Wang, X., Goldschlag, Y., Gaur, Y., Babaei, Y., Wen, Y., Song, Y., Zhang, Y., Li, Y., Mao, Y., Coudert, Z. D., Yan, Z., Chen, Z., Papakipos, Z., Singh, A., Grattafiori, A., Jain, A., Kelsey, A., Shajnfeld, A., Gangidi, A., Victoria, A., Goldstand, A., Menon, A., Sharma, A., Boesenberg, A., Vaughan, A., Baevski, A., Feinstein, A., Kallet, A., Sangani, A., Yunus, A., Lupu, A., Alvarado, A., Caples, A., Gu, A., Ho, A., Poulton, A., Ryan, A., Ramchandani, A., Franco, A., Saraf, A., Chowdhury, A., Gabriel, A., Bharambe, A., Eisenman, A., Yazdan, A., James, B., Maurer, B., Leonhardi, B., Huang, B., Loyd, B., Paola, B. D., Paranjape, B., Liu, B., Wu, B., Ni, B., Hancock,

B., Wasti, B., Spence, B., Stojkovic, B., Gamido, B., Montalvo, B., Parker, C., Burton, C., Mejia, C., Wang, C., Kim, C., Zhou, C., Hu, C., Chu, C.-H., Cai, C., Tindal, C., Feichtenhofer, C., Civin, D., Beaty, D., Kreymer, D., Li, D., Wyatt, D., Adkins, D., Xu, D., Testuggine, D., David, D., Parikh, D., Liskovich, D., Foss, D., Wang, D., Le, D., Holland, D., Dowling, E., Jamil, E., Montgomery, E., Presani, E., Hahn, E., Wood, E., Brinkman, E., Arcaute, E., Dunbar, E., Smothers, E., Sun, F., Kreuk, F., Tian, F., Ozgenel, F., Caggioni, F., Guzmán, F., Kanayet, F., Seide, F., Florez, G. M., Schwarz, G., Badeer, G., Swee, G., Halpern, G., Thattai, G., Herman, G., Sizov, G., Guangyi, Zhang, Lakshminarayanan, G., Shojanazeri, H., Zou, H., Wang, H., Zha, H., Habeeb, H., Rudolph, H., Suk, H., Aspegren, H., Goldman, H., Damla, I., Molybog, I., Tufanov, I., Veliche, I.-E., Gat, I., Weissman, J., Geboski, J., Kohli, J., Asher, J., Gaya, J.-B., Marcus, J., Tang, J., Chan, J., Zhen, J., Reizenstein, J., Teboul, J., Zhong, J., Jin, J., Yang, J., Cummings, J., Carvill, J., Shepard, J., McPhie, J., Torres, J., Ginsburg, J., Wang, J., Wu, K., U, K. H., Saxena, K., Prasad, K., Khandelwal, K., Zand, K., Matosich, K., Veeraraghavan, K., Michelena, K., Li, K., Huang, K., Chawla, K., Lakhota, K., Huang, K., Chen, L., Garg, L., A, L., Silva, L., Bell, L., Zhang, L., Guo, L., Yu, L., Moshkovich, L., Wehrstedt, L., Khabsa, M., Avalani, M., Bhatt, M., Tsimpoukelli, M., Mankus, M., Hasson, M., Lennie, M., Reso, M., Groshev, M., Naumov, M., Lathi, M., Keneally, M., Seltzer, M. L., Valko, M., Restrepo, M., Patel, M., Vyatskov, M., Samvelyan, M., Clark, M., Macey, M., Wang, M., Hermoso, M. J., Metanat, M., Rastegari, M., Bansal, M., Santhanam, N., Parks, N., White, N., Bawa, N., Singhal, N., Egebo, N., Usunier, N., Laptev, N. P., Dong, N., Zhang, N., Cheng, N., Chernoguz, O., Hart, O., Salpekar, O., Kalinli, O., Kent, P., Parekh, P., Saab, P., Balaji, P., Rittner, P., Bontrager, P., Roux, P., Dollar, P., Zvyagina, P., Ratanchandani, P., Yuvraj, P., Liang, Q., Alao, R., Rodriguez, R., Ayub, R., Murthy, R., Nayani, R., Mitra, R., Li, R., Hogan, R., Battey, R., Wang, R., Maheswari, R., Howes, R., Rinott, R., Bondu, S. J., Datta, S., Chugh, S., Hunt, S., Dhillon, S., Sidorov, S., Pan, S., Verma, S., Yamamoto, S., Ramaswamy, S., Lindsay, S., Lindsay, S., Feng, S., Lin, S., Zha, S. C., Shankar, S., Zhang, S., Zhang, S., Wang, S., Agarwal, S., Sajuyigbe, S., Chintala, S., Max, S., Chen, S., Kehoe, S., Satterfield, S., Govindaprasad, S., Gupta, S., Cho, S., Virk, S., Subramanian, S., Choudhury, S., Goldman, S., Remez, T., Glaser, T., Best, T., Kohler, T., Robinson, T., Li, T., Zhang, T., Matthews, T.,

- Chou, T., Shaked, T., Vontimitta, V., Ajayi, V., Montanez, V., Mohan, V., Kumar, V. S., Mangla, V., Albiero, V., Ionescu, V., Poenaru, V., Mihailescu, V. T., Ivanov, V., Li, W., Wang, W., Jiang, W., Bouaziz, W., Constable, W., Tang, X., Wang, X., Wu, X., Wang, X., Xia, X., Wu, X., Gao, X., Chen, Y., Hu, Y., Jia, Y., Qi, Y., Li, Y., Zhang, Y., Zhang, Y., Adi, Y., Nam, Y., Yu, Wang, Hao, Y., Qian, Y., He, Y., Rait, Z., DeVito, Z., Rosnbrick, Z., Wen, Z., Yang, Z., and Zhao, Z. (2024). The llama 3 herd of models.
- Emonet, V., Bolleman, J., Duvaud, S., de Farias, T. M., and Sima, A. C. (2024). Llm-based sparql query generation from natural language over federated knowledge graphs.
- Fu, B., Qiu, Y., Tang, C., Li, Y., Yu, H., and Sun, J. (2020). A survey on complex question answering over knowledge base: Recent advances and challenges.
- Gao, Y., Xiong, Y., Gao, X., Jia, K., Pan, J., Bi, Y., Dai, Y., Sun, J., Wang, M., and Wang, H. (2024). Retrieval-augmented generation for large language models: A survey.
- Gosnell, D. and de Saint Pern, V. (2024). Improving retrieval-augmented generation accuracy with graphrag. Accessed: 2024-12-29.
- Graves, A. (2014). Generating sequences with recurrent neural networks.
- Hao, G., Wu, J., Pan, Q., and Morello, R. (2024). Quantifying the uncertainty of llm hallucination spreading in complex adaptive social networks. *Scientific Reports (Nature Publisher Group)*, 14(1):16375. Copyright - The Author(s) 2024. This work is published under <http://creativecommons.org/licenses/by/4.0/> (the License). Notwithstanding the ProQuest Terms and Conditions, you may use this content in accordance with the terms of the License; Last updated - 2024-07-17.
- Harth, A. et al., editors (2014). *Linked Data Management*. CRC Press LLC. ProQuest Ebook Central.
- He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep residual learning for image recognition.
- Hevner, A. R., March, S. T., Park, J., and Ram, S. (2004). Design science in information systems research. *MIS quarterly*, pages 75–105.
- Hochreiter, S., Bengio, Y., Frasconi, P., Schmidhuber, J., et al. (2001). Gradient flow in recurrent nets: the difficulty of learning long-term dependencies.
- Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8):1735–1780.
- Hosseini, P., Castro, I., Ghinassi, I., and Purver, M. (2024). Efficient solutions for an

intriguing failure of llms: Long context window does not mean llms can analyze long sequences flawlessly.

- Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., and Chen, W. (2021). Lora: Low-rank adaptation of large language models.
- Hui, B., Yang, J., Cui, Z., Yang, J., Liu, D., Zhang, L., Liu, T., Zhang, J., Yu, B., Lu, K., Dang, K., Fan, Y., Zhang, Y., Yang, A., Men, R., Huang, F., Zheng, B., Miao, Y., Quan, S., Feng, Y., Ren, X., Ren, X., Zhou, J., and Lin, J. (2024). Qwen2.5-coder technical report.
- Jiang, A. Q., Sablayrolles, A., Mensch, A., Bamford, C., Chaplot, D. S., de las Casas, D., Bressand, F., Lengyel, G., Lample, G., Saulnier, L., Lavaud, L. R., Lachaux, M.-A., Stock, P., Scao, T. L., Lavril, T., Wang, T., Lacroix, T., and Sayed, W. E. (2023). Mistral 7b.
- Jiang, H., Wu, Q., Luo, X., Li, D., Lin, C.-Y., Yang, Y., and Qiu, L. (2024). LongLLMLingua: Accelerating and enhancing LLMs in long context scenarios via prompt compression. In Ku, L.-W., Martins, A., and Srikumar, V., editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1658–1677, Bangkok, Thailand. Association for Computational Linguistics.
- JSON (n.d.). Introducing json. Accessed: 2024-07-13.
- Juneja, D. and Sharma, M. K. (2017). An approach to understand semantic web. *International Journal of Computer Applications*, 167(11):36–39. Accessed: 2024-07-17.
- Karpukhin, V., Ouz, B., Min, S., Lewis, P., Wu, L., Edunov, S., Chen, D., and tau Yih, W. (2020). Dense passage retrieval for open-domain question answering.
- Kaushik, P., Gain, A., Kortylewski, A., and Yuille, A. (2021). Understanding catastrophic forgetting and remembering in continual learning with optimal relevance mapping.
- Kifer, M. and Boley, H. (2013). Rif overview (second edition). Accessed: 2024-06-23.
- Kim, J., Kwon, Y., Jo, Y., and Choi, E. (2023). KG-GPT: A general framework for reasoning on knowledge graphs using large language models. In Bouamor, H., Pino, J., and Bali, K., editors, *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 9410–9421, Singapore. Association for Computational Linguistics.
- Kovriguina, L., Teucher, R., Radyush, D., and Mouromtsev, D. (2023). Sparqlgen:

- One-shot prompt-based approach for sparql query generation. In *International Conference on Semantic Systems*.
- Kublik, S. and Saboo, S. (2022). *GPT-3: Building Innovative NLP Products Using Large Language Models*. O'Reilly Media, Incorporated.
- Kusupati, A., Bhatt, G., Rege, A., Wallingford, M., Sinha, A., Ramanujan, V., Howard-Snyder, W., Chen, K., Kakade, S., Jain, P., and Farhadi, A. (2024). Matryoshka representation learning. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA. Curran Associates Inc.
- Lai, G., Xie, Q., Liu, H., Yang, Y., and Hovy, E. (2017). RACE: Large-scale ReAding comprehension dataset from examinations. In Palmer, M., Hwa, R., and Riedel, S., editors, *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 785–794, Copenhagen, Denmark. Association for Computational Linguistics.
- Larson, J. and Truitt, S. (2024). Graphrag: A new approach for discovery using complex information. Accessed: 22 July 2024.
- Lewis, M., Liu, Y., Goyal, N., Ghazvininejad, M., Mohamed, A., Levy, O., Stoyanov, V., and Zettlemoyer, L. (2019). Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension.
- Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Kttler, H., Lewis, M., tau Yih, W., Rocktschel, T., Riedel, S., and Kiela, D. (2021). Retrieval-augmented generation for knowledge-intensive nlp tasks.
- Liu, C., Wang, S., Kang, Y., Qing, L., Zhao, F., Sun, C., Kuang, K., and Wu, F. (2024). More than catastrophic forgetting: Integrating general capabilities for domain-specific llms.
- Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. (2019). Roberta: A robustly optimized bert pretraining approach.
- Maas, A. L., Daly, R. E., Pham, P. T., Huang, D., Ng, A. Y., and Potts, C. (2011). Learning word vectors for sentiment analysis. In Lin, D., Matsumoto, Y., and Mihalcea, R., editors, *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA. Association for Computational Linguistics.

- Mandolang, D. C. (2024). curriculum-knowledge-graph. <https://github.com/danielcm585/curriculum-knowledge-graph>.
- Mavromatis, C. and Karypis, G. (2024). Gnn-rag: Graph neural retrieval for large language model reasoning.
- Meindertma, J. (2019). What's the best rdf serialization format? Accessed: 2024-07-13.
- Mistral AI Team (2024). Mistral nemo. <https://mistral.ai/news/mistral-nemo/>. Accessed: 2024-12-03.
- Mozilla Developer Network (n.d.). Working with json. Accessed: 2024-07-13.
- Obitko, M. (2007). Semantic web architecture. Accessed: 2024-07-17.
- Ontotext (n.d.). What are ontologies? Accessed: 2024-07-13.
- OpenAI (2023). Gpt-4 technical report.
- Ovadia, O., Brief, M., Mishaeli, M., and Elisha, O. (2024). Fine-tuning or retrieval? comparing knowledge injection in llms.
- Peng, B., Zhu, Y., Liu, Y., Bo, X., Shi, H., Hong, C., Zhang, Y., and Tang, S. (2024). Graph retrieval-augmented generation: A survey.
- Perevalov, A., Diefenbach, D., Usbeck, R., and Both, A. (2022). Qald-9-plus: A multilingual dataset for question answering over dbpedia and wikidata translated by native speakers. In *2022 IEEE 16th International Conference on Semantic Computing (ICSC)*, pages 229–234.
- Qwen Team (2024). Qwen2.5: A party of foundation models. <https://qwenlm.github.io/blog/qwen2.5/>. Accessed: 2024-12-05.
- Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. (2019). Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.
- Rafailov, R., Sharma, A., Mitchell, E., Ermon, S., Manning, C. D., and Finn, C. (2024). Direct preference optimization: your language model is secretly a reward model. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA. Curran Associates Inc.
- Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. (2019). Exploring the limits of transfer learning with a unified text-to-text transformer.
- Rajpurkar, P., Zhang, J., Lopyrev, K., and Liang, P. (2016). SQuAD: 100,000+ questions for machine comprehension of text. In Su, J., Duh, K., and Carreras, X., editors, *Proceedings of the 2016 Conference on Empirical Methods in Natural Language*

- Processing*, pages 2383–2392, Austin, Texas. Association for Computational Linguistics.
- Raschka, S. (2024). *Build a Large Language Model (From Scratch)*. From Scratch. Manning.
- Reimers, N. and Gurevych, I. (2019). Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.
- Roberts, A., Raffel, C., and Shazeer, N. (2020). How much knowledge can you pack into the parameters of a language model?
- Robertson, S. and Zaragoza, H. (2009). The probabilistic relevance framework: Bm25 and beyond. *Found. Trends Inf. Retr.*, 3(4):333389.
- Rony, M. R. A. H., Kumar, U., Teucher, R., Kovriguina, L., and Lehmann, J. (2022). Sgpt: A generative approach for sparql query generation from natural language questions. *IEEE Access*, 10:70712–70723.
- Sarmah, B., Hall, B., Rao, R., Patel, S., Pasquali, S., and Mehta, D. (2024). Hybridrag: Integrating knowledge graphs and vector retrieval augmented generation for efficient information extraction.
- Schmid, P., Sanseviero, O., Bartolome, A., von Werra, L., Vila, D., Srivastav, V., Sun, M., and Cuenca, P. (2024). Llama 3.1 - 405b, 70b & 8b with multilinguality and long context. Accessed: 2024-12-29.
- Singhal, A. (2012). Introducing the knowledge graph: Things, not strings.
- Song, K., Tan, X., Qin, T., Lu, J., and Liu, T.-Y. (2020). Mpnnet: masked and permuted pre-training for language understanding. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA. Curran Associates Inc.
- Sturua, S., Mohr, I., Akram, M. K., Gnther, M., Wang, B., Krimmel, M., Wang, F., Mastrapas, G., Koukounas, A., Wang, N., and Xiao, H. (2024). jina-embeddings-v3: Multilingual embeddings with task lora.
- Su, J., Ahmed, M., Lu, Y., Pan, S., Bo, W., and Liu, Y. (2024). Roformer: Enhanced transformer with rotary position embedding. *Neurocomput.*, 568(C).
- Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks.
- Taunk, D., Khanna, L., Kandru, S. V. P. K., Varma, V., Sharma, C., and Tapaswi, M.

- (2023). Grapeqa: Graph augmentation and pruning to enhance question-answering. In *Companion Proceedings of the ACM Web Conference 2023*, WWW '23 Companion, page 11381144, New York, NY, USA. Association for Computing Machinery.
- The Unicode Consortium (2023). The unicode standard, version 15.1.0. Accessed: 2024-06-14.
- Tuominen, J. (2020). Owl: Web ontology language. <http://heldig.fi>. Presented at CS-E4410 Semantic Web, Aalto University, February 5, 2020.
- Usbeck, R., Gusmita, R. H., Ngomo, A.-C. N., and Saleem, M. (2018). 9th challenge on question answering over linked data (qald-9) (invited paper). In *Semdeep/NLIWoD@ISWC*.
- Vanderbilt Libraries (2019). Linked data basics: Rdf serializations and triplestores. Accessed: 2024-07-18.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. (2017). Attention is all you need. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.
- Verborgh, R., Usbeck, R., Röder, M., Hoffmann, M., Conrads, F., Huthmann, J., Ngonga-Ngomo, A.-C., Demmler, C., Unger, C., Ngonga Ngomo, A.-C., Fundulaki, I., and Krithara, A. (2019). Benchmarking question answering systems. *Semant. Web*, 10(2):293–304.
- vom Brocke, J., Hevner, A., and Maedche, A. (2020). *Introduction to Design Science Research*, pages 1–13. Springer International Publishing, Cham.
- von Eschenbach, W. J. (2021). Transparency and the black box problem: Why we do not trust ai. *Philosophy & Technology*, 34(4):1607–1622. Copyright - The Author(s), under exclusive licence to Springer Nature B.V. 2021; Last updated - 2023-12-03.
- Vrandečić, D. (2023). The future of knowledge graphs in a world of large language models.
- Vrandečić, D. and Krötzsch, M. (2014). Wikidata: a free collaborative knowledgebase. *Commun. ACM*, 57(10):7885.
- W3C (2014). RDF Schema 1.1. Accessed: 2024-06-22.
- W3Techs (2024). Usage statistics of json-ld for websites. Accessed: 2024-07-13.

- Waagmeester, A., Stupp, G., Burgstaller-Muehlbacher, S., Good, B. M., Griffith, M., Griffith, O. L., Hanspers, K., Hermjakob, H., Hudson, T. S., Hybiske, K., Keating, S. M., Manske, M., Mayers, M., Mietchen, D., Mitraka, E., Pico, A. R., Putman, T., Riutta, A., Queralt-Rosinach, N., Schriml, L. M., Shafee, T., Slenter, D., Stephan, R., Thornton, K., Tsueng, G., Tu, R., Ul-Hasan, S., Willighagen, E., Wu, C., and Su, A. I. (2020). Science forum: Wikidata as a knowledge graph for the life sciences. *eLife*, 9:e52614.
- Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. (2018). GLUE: A multi-task benchmark and analysis platform for natural language understanding. In Linzen, T., Chrupała, G., and Alishahi, A., editors, *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355, Brussels, Belgium. Association for Computational Linguistics.
- Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E. H., Le, Q. V., and Zhou, D. (2024a). Chain-of-thought prompting elicits reasoning in large language models. In *Proceedings of the 36th International Conference on Neural Information Processing Systems, NIPS '22*, Red Hook, NY, USA. Curran Associates Inc.
- Wei, J., Yao, Y., Jean-Francois, T., Guo, H., Estornell, A., and Liu, Y. (2024b). Measuring and reducing llm hallucination without gold-standard answers. Copyright - 2024. This work is published under <http://arxiv.org/licenses/nonexclusive-distrib/1.0/> (the License). Notwithstanding the ProQuest Terms and Conditions, you may use this content in accordance with the terms of the License; Last updated - 2024-06-11.
- Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R., and Le, Q. V. (2020). Xlnet: Generalized autoregressive pretraining for language understanding.
- Yani, M. and Krisnadhi, A. A. (2021). Challenges, techniques, and trends of simple knowledge graph question answering: A survey. *Information*, 12(7).
- Yasunaga, M., Ren, H., Bosselut, A., Liang, P., and Leskovec, J. (2021). QA-GNN: Reasoning with language models and knowledge graphs for question answering. In Toutanova, K., Rumshisky, A., Zettlemoyer, L., Hakkani-Tur, D., Beltagy, I., Bethard, S., Cotterell, R., Chakraborty, T., and Zhou, Y., editors, *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 535–546,

Online. Association for Computational Linguistics.

- Zaratiana, U., Tomeh, N., Holat, P., and Charnois, T. (2023). Gliner: Generalist model for named entity recognition using bidirectional transformer.
- Zhang, J., Zhang, X., Yu, J., Tang, J., Tang, J., Li, C., and Chen, H. (2022). Subgraph retrieval enhanced model for multi-hop knowledge base question answering. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics.
- Zhu, Y., Yuan, H., Wang, S., Liu, J., Liu, W., Deng, C., Chen, H., Liu, Z., Dou, Z., and Wen, J.-R. (2024). Large language models for information retrieval: A survey.
- Ziegler, D. M., Stiennon, N., Wu, J., Brown, T. B., Radford, A., Amodei, D., Christiano, P., and Irving, G. (2020). Fine-tuning language models from human preferences.



# APPENDIX



## APPENDIX 1: LOCAL KG DATASET GENERATOR

### Code for Local KG Dataset Generator

```
1 import warnings
2 from util import get_next_variable, is_dbpedia_entity_iri, is_wikidata_entity_iri,
3 replace_prefix_dbpedia, replace_prefix_wikidata
4 from llm import chat_model
5 from typing import List
6 from rdflib import Graph
7 from rdflib.namespace import RDF
8 import random
9 from rdflib import Literal
10 import re
11 from tqdm import tqdm
12 from timeout import timeout
13 import pandas as pd
14 import os
15
16 warnings.filterwarnings("ignore")
17
18 class QADatasetGenerator:
19 def __init__(self, source: str, excluded_props: List[str], timeout = 40):
20 self.source = source
21 self.excluded_props = excluded_props
22 self.timeout = timeout
23 self.graph = None
24 self.graph = Graph()
25 self.graph.parse(source)
26
27 def write_to_file(self, dataset_name: str, amount: int, category: str, count: bool):
28 # count can be used with simple only
29 if count and category.startswith("complex"):
30 raise ValueError("Count for complex queries is not supported")
31
32 questions, queries = self.generate(amount, category, count)
33 df = {
34 "question": questions,
35 "query": queries
36 }
37 df = pd.DataFrame(df)
38 status = "count" if count else "normal"
39
40 directory = f'dataset/io/{dataset_name}'
41 if not os.path.exists(directory):
42 os.makedirs(directory)
43
44 df.to_json(f'{directory}/{category}_{amount}_{status}.json', orient='records',
45 indent=4)
46 print("Finished writing dataset")
```

```

47
48
49 def generate(self, amount: int, category: str, count: bool):
50 questions = []
51 queries = []
52 for _ in tqdm(range(amount)):
53 while True:
54 try:
55 with timeout(self.timeout):
56 cat = category.split("_")
57 question, query = "", ""
58 if cat[0] == "simple":
59 if count:
60 question, query = self.generate_count(cat[1])
61 else:
62 question, query = self.generate_simple(cat[1])
63 break
64 elif cat[0] == "complex":
65 question, query = self.generate_complex(cat[1])
66 break
67 if query in queries:
68 raise ValueError()
69 except TimeoutError:
70 print("Timeout, repeating")
71 continue
72 except ValueError:
73 print("Duplicate query, repeating")
74 continue
75 except Exception as e:
76 print(f"Error: {e}")
77 continue
78 question, query = question.strip(), query.strip()
79 questions.append(question)
80 queries.append(query)
81 return questions, queries
82
83 def __refine_question(self, mapping, query):
84 mapping_in_sentence = ""
85 for uri, label in mapping.items():
86 if "dbpedia" in self.source:
87 pref = replace_prefix_dbpedia(uri)
88 if pref in query:
89 mapping_in_sentence += f"{pref} has human-readable name '{label}'\n"
90 elif "wikidata" in self.source:
91 pref = replace_prefix_wikidata(uri)
92 if pref in query:
93 mapping_in_sentence += f"{pref} has human-readable name '{label}'\n"
94 else:
95 if uri in query:
96 mapping_in_sentence += f"{uri} has human-readable name '{label}'\n"

```

```

97
98 prompt = f"""Having a SPARQL query:
99 {query}
100 Where:
101 {mapping_in_sentence}
102 Transform the SPARQL query to a natural language question.
103 Output just the transformed question
104 """
105 print(prompt)
106 result = chat_model.invoke(prompt).content
107 if "Here" in result:
108 matched = re.search(r'"(.*)"', result, re.DOTALL)
109 if matched:
110 return matched.group(1)
111 else:
112 return result
113 return result
114
115 def __get_label(self, entity):
116 try:
117 query = f"""
118 select ?lit {{
119 <{entity}> rdfs:label ?lit .
120 }}
121 """
122 result = list(self.graph.query(query))[0][0].toPython()
123 return result
124 except:
125 # if literal
126 result = entity.toPython()
127 if type(result) is str:
128 return result
129 elif type(result) is int:
130 return f"{result}^^xsd:integer"
131
132 def __filter_prop_query(self):
133 _filter = [f"contains(str(?p), '{uri}')" = false" for uri in self.excluded_props]
134 return " && ".join(_filter)
135
136 def __random_walk(self, entity):
137 filter_prop = self.__filter_prop_query()
138 query = f"""
139 select ?p ?o {{
140 <{entity}> ?p ?o .
141 filter (
142 {filter_prop}
143)
144 }}
145 """
146 res = list(self.graph.query(query))

```

```

147 choice = list(random.choice(res))
148 choice.insert(0, entity)
149 return tuple(choice)
150
151 def __get_one_triple(self, subject = None):
152 start_given = subject != None
153 err = True
154 while err:
155 try:
156 if not start_given:
157 subject = self.__random_pick_entity()
158 triple = self.__random_walk(subject)
159 err = False
160 except Exception as e:
161 pass
162 return triple
163
164 def __concat_str_with_datatype(self, prop, o):
165 query = f"select ?range {{ <{prop}> rdfs:range ?range . }}"
166 mapping = {
167 "http://www.w3.org/2001/XMLSchema#": "xsd:",
168 "http://dbpedia.org/datatype/": "dbd:"
169 }
170 datatype = list(self.graph.query(query))[0][0]
171 for k, v in mapping.items():
172 tmp = datatype.replace(k, v)
173 if tmp == "xsd:string":
174 return f"'{o}'"
175 datatype = tmp
176 return f"'{o}'^^{datatype}"
177
178 def __is_no_property(self, entity):
179 if isinstance(entity, Literal):
180 return True
181 try:
182 self.__random_walk(entity)
183 return False
184 except IndexError:
185 return True
186
187 def generate_count(self, category):
188 # this uses simple pattern only
189 mapping, answer = self.generate_simple(category, return_question=False)
190 new_answer = answer.replace("?x", "(count(?x) as ?cnt)", 1)
191 question = self.__refine_question(mapping, new_answer)
192 return question, new_answer
193
194 def generate_simple(self, category, return_question = True):
195 # one triple pattern
196 # supports only a b ?x

```

```

197 triple = self.__get_one_triple()
198 query_uri = "select ?x {{ <{s}> <{p}> ?x . }}"
199 query_uri_reverse = "select ?x {{ ?x <{p}> {o} . }}"
200
201 if not isinstance(triple[2], Literal):
202 s, p, o = self.__get_label(triple[0]), self.__get_label(triple[1]),
203 self.__get_label(triple[2])
204 else:
205 s, p, o = self.__get_label(triple[0]), self.__get_label(triple[1]),
206 triple[2].toPython()
207 mapping = {triple[0]: s, triple[1]: p, triple[2]: o}
208 if category == "1":
209 answer = query_uri.format(s=triple[0], p=triple[1], o=triple[2])
210 else:
211 if not isinstance(triple[2], Literal):
212 answer = query_uri_reverse.format(s=triple[0], p=triple[1], o=f"<{triple[2]}>")
213 else:
214 answer = query_uri_reverse.format(s=triple[0], p=triple[1], o=
215 f"{self.__concat_str_with_datatype(triple[1], triple[2])}")
216 if return_question:
217 refined_question = self.__refine_question(mapping, answer)
218 return refined_question, answer
219 else:
220 return mapping, answer
221
222 def generate_complex(self, category, max_triples = 3):
223 starting_triple = self.__get_one_triple()
224 depth = random.choice([i for i in range(2, max_triples)])
225
226 if category == '1':
227 # pattern: ?x y z ; a b .
228 subject = starting_triple[0]
229 triples = set()
230 triples.add(starting_triple)
231 while len(triples) <= depth:
232 triple = self.__get_one_triple(subject)
233 triples.add(triple)
234
235 triple_pattern = []
236 for (_, p, o) in triples:
237 if isinstance(o, Literal):
238 result = o.toPython()
239 if type(result) is str:
240 triple_pattern.append(f"?x <{p}> '{o}'")
241 elif type(result) is int:
242 triple_pattern.append(f"?x <{p}> '{o}'^^xsd:integer")
243 else:
244 triple_pattern.append(f"?x <{p}> <{o}>")
245
246 triple_pattern = " . ".join(triple_pattern) + " ."

```

```

247 query = f"select ?x {{ {triple_pattern} }}"
248 mapping = {}
249 for (_, p, o) in triples:
250 if is_wikidata_entity_iri(o) or is_dbpedia_entity_iri(o):
251 p_label, o_label = self.__get_label(p), self.__get_label(o)
252 else:
253 p_label, o_label = self.__get_label(p), o
254 mapping[p] = p_label
255 mapping[o] = o_label
256 refined_question = self.__refine_question(mapping, query)
257 return refined_question, query
258
259 elif category == '2':
260 # pattern: ?x a ?y . ?y b c .
261 # make sure to prevent object as literal and make sure the object has at least
262 # one property excluding the properties mentioned in exclude list
263 # kadang ada yg tidak ketemu match, harus repeat
264 # search such that the first triple is not literal as the object
265 while self.__is_no_property(starting_triple[2]):
266 starting_triple = self.__get_one_triple()
267 triples = []
268 triples.append(starting_triple)
269
270 triple = starting_triple
271 while len(triples) < depth and not isinstance(triple[2], Literal):
272 triple = self.__get_one_triple(triples[-1][2])
273 triples.append(triple)
274
275 triple_pattern = []
276 curr_var = "x"
277 for i in range(len(triples)):
278 p = triples[i][1]
279 o = triples[i][2]
280 if i == len(triples) - 1:
281 if isinstance(o, Literal):
282 result = o.toPython()
283 if isinstance(result, str):
284 triple_pattern.append(f"?{curr_var} <{p}> '{o}'")
285 elif isinstance(result, int):
286 triple_pattern.append(f"?{curr_var} <{p}> '{o}'^^xsd:integer")
287 else:
288 triple_pattern.append(f"?{curr_var} <{p}> <{o}>")
289 else:
290 triple_pattern.append(f"?{curr_var} <{p}> ?{get_next_variable(curr_var)}")
291 curr_var = get_next_variable(curr_var)
292 triple_pattern = " . ".join(triple_pattern) + " ."
293
294 query = f"select ?x {{ {triple_pattern} }}"
295 mapping = {}
296 for i in range(len(triples)):

```

```
297 s, p, o = triples[i]
298 p_label, o_label = self.__get_label(p), self.__get_label(o)
299 mapping[p] = p_label
300 mapping[o] = o_label
301 refined_question = self.__refine_question(mapping, query)
302 return refined_question, query
303
304 def __random_pick_entity(self):
305 candidates = set()
306 for (s, p, _) in self.graph:
307 if p == RDF['type']:
308 candidates.add(s)
309 entity = random.choice(list(candidates))
310 return entity
```

**Code 1:** Code for Local KG Dataset Generator

## APPENDIX 2: QALD TEST DATASET

### JSON of QALD Test Dataset

```
1 {
2 "query_name": {
3 "0": "Give me all movies with Tom Cruise.",
4 "1": "Which poet wrote the most books?",
5 "2": "In which countries can you pay using the West African CFA franc?",
6 "3": "How many people live in the capital of Australia?",
7 "4": "how big is the total area of North Rhine- Westphalia?",
8 "5": "What is the total population of Melbourne, Florida?",
9 "6": "List all games by GMT.",
10 "7": "How much did the Lego Movie cost?",
11 "8": "Give me all the TV shows with Neil Patrick Harris.",
12 "9": "In which country is the Limerick Lake?",
13 "10": "How tall is Michael Jordan?",
14 "11": "In which programming language is GIMP written?",
15 "12": "How many children does Eddie Murphy have?",
16 "13": "Give me the currency of China.",
17 "14": "How many languages are spoken in Colombia?",
18 "15": "Give me all Danish films.",
19 "16": "In which ancient empire could you pay with cocoa beans?",
20 "17": "How often did Jane Fonda marry?",
21 "18": "How many politicians graduated from Columbia University?",
22 "19": "Show me Hemingway's autobiography.",
23 "20": "Give me all soccer clubs in Spain.",
24 "21": "how much is the total population of european union?",
25 "22": "How many employees does IBM have?",
26 "23": "Which country has the most official languages?",
27 "24": "Give me all breeds of the German Shepherd dog.",
28 "25": "Show me all books in Asimov's Foundation series.",
29 "26": "In which year was Rachel Stevens born?",
30 "27": "how much is the population Iraq?",
31 "28": "Which countries have more than two official languages?",
32 "29": "How much did Pulp Fiction cost?",
33 "30": "Which actor was casted in the most movies?",
34 "31": "What is the most frequent cause of death?",
35 "32": "List all boardgames by GMT.",
36 "33": "How many programming languages are there?",
37 "34": "Give me all Australian metalcore bands.",
38 "35": "Who is the author of the interpretation of dreams?",
39 "36": "Which musician wrote the most books?",
40 "37": "How many children did Benjamin Franklin have?",
41 "38": "To which artistic movement did the painter of The Three Dancers belong?",
42 "39": "Which Greek parties are pro-European?",
43 "40": "How many films did Hal Roach produce?",
44 "41": "Give me all companies in Munich.",
45 "42": "Show a list of soccer clubs that play in the Bundesliga.",
46 "43": "Give me the runtime of Toy Story.",
```

```

47 "44": "How many missions does the Soyuz programme have?",
48 "45": "Which holidays are celebrated around the world? test2",
49 "46": "How much is the population of mexico city ?",
50 "47": "How many airlines are members of the Star Alliance?",
51 "48": "How many people live in Eurasia?",
52 "49": "how much is the elevation of D\u00fcrsseldorf Airport ?",
53 "50": "List all the musicals with music by Elton John.",
54 "51": "In which films directed by Garry Marshall was Julia Roberts starring?",
55 "52": "How many languages are spoken in Turkmenistan?",
56 "53": "Give me all libraries established earlier than 1400.",
57 "54": "Who developed Slack?",
58 "55": "Give me all launch pads operated by NASA.",
59 "56": "How many states are in Mexico?",
60 "57": "Show me all the breweries in Australia.",
61 "58": "Which telecommunications organizations are located in Belgium?",
62 "59": "How many theories did Albert Einstein come up with?",
63 "60": "What was Brazil's lowest rank in the FIFA World Ranking?",
64 "61": "Which programming languages influenced Javascript?",
65 "62": "Show me all songs from Bruce Springsteen released between 1980 and 1990.",
66 "63": "How many airlines are there?",
67 "64": "Give me all films produced by Hal Roach."
68 },
69 "dbpedia_query": {
70 "0": "SELECT DISTINCT ?uri WHERE { ?uri a dbo:Film ; dbo:starring dbr:Tom_Cruise }",
71 "1": "SELECT DISTINCT ?uri WHERE { ?uri dbo:occupation dbr:Poet . ?x dbo:author ?uri .
72 ?x rdf:type dbo:Book . } ORDER BY DESC(COUNT(?x)) OFFSET 0 LIMIT 1",
73 "2": "SELECT DISTINCT ?uri WHERE { ?uri dbo:currency dbr:West_African_CFA_franc }",
74 "3": "SELECT DISTINCT ?num WHERE { dbr:Australia dbo:capital ?x .
75 ?x dbo:populationTotal ?num }",
76 "4": "SELECT ?tarea WHERE { dbr:North_Rhine-Westphalia dbo:areaTotal ?tarea }",
77 "5": "SELECT ?uri WHERE { <http://dbpedia.org/resource/Melbourne,_Florida>
78 dbo:populationTotal ?uri }",
79 "6": "SELECT DISTINCT ?uri WHERE { ?uri dbo:publisher dbr:GMT_Games }",
80 "7": "SELECT DISTINCT ?n WHERE { dbr:The_Lego_Movie dbo:budget ?n }",
81 "8": "SELECT DISTINCT ?uri WHERE { ?uri a dbo:TelevisionShow ;
82 dbo:starring dbr:Neil_Patrick_Harris }",
83 "9": "SELECT DISTINCT ?uri WHERE { dbr:Limerick_Lake dbo:country ?uri }",
84 "10": "SELECT DISTINCT ?num WHERE { dbr:Michael_Jordan dbo:height ?num }",
85 "11": "SELECT DISTINCT ?uri WHERE { dbr:GIMP dbo:programmingLanguage ?uri }",
86 "12": "SELECT DISTINCT ?n WHERE { dbr:Eddie_Murphy
87 <http://dbpedia.org/property/children> ?n }",
88 "13": "SELECT DISTINCT ?uri WHERE { dbr:China dbo:currency ?uri }",
89 "14": "SELECT (COUNT(DISTINCT ?uri) AS ?c) WHERE
90 { ?uri rdf:type dbo:Language . dbr:Colombia dbo:language ?uri }",
91 "15": "SELECT DISTINCT ?uri WHERE { ?uri rdf:type dbo:Film ;
92 dbo:country dbr:Denmark }",
93 "16": "SELECT DISTINCT ?uri WHERE { ?uri dbo:currency dbr:Cocoa_bean }",
94 "17": "SELECT (COUNT(DISTINCT ?uri) AS ?c) WHERE { dbr:Jane_Fonda dbo:spouse ?uri }",
95 "18": "SELECT (COUNT(DISTINCT ?uri) AS ?c) WHERE { ?uri dbo:profession dbr:Politician ;
96 dbo:almaMater dbr:Columbia_University }",

```

```

97 "19": "SELECT DISTINCT ?uri WHERE { ?uri dbo:author dbr:Ernest_Hemingway ;
98 dbo:literaryGenre dbr:Autobiography }",
99 "20": "SELECT DISTINCT ?uri WHERE { ?uri a dbo:SoccerClub { ?uri dbo:ground dbr:Spain }
100 UNION { ?uri <http://dbpedia.org/property/ground> ?ground
101 FILTER regex(?ground, \"Spain\") } }",
102 "21": "SELECT ?europop WHERE { dbr:European_Union dbo:populationTotal ?europop }",
103 "22": "SELECT DISTINCT ?number WHERE { dbr:IBM dbo:numberOfEmployees ?number }",
104 "23": "SELECT DISTINCT ?uri WHERE { ?uri rdf:type dbo:Country .
105 ?uri <http://dbpedia.org/property/officialLanguages> ?language . }
106 ORDER BY DESC(COUNT(?language)) OFFSET 0 LIMIT 1",
107 "24": "SELECT DISTINCT ?uri WHERE { ?uri
108 <http://dbpedia.org/property/breed> dbr:German_Shepherd }",
109 "25": "SELECT DISTINCT ?uri WHERE { ?uri dbo:series dbr:Foundation_series }",
110 "26": "SELECT DISTINCT ?uri WHERE { dbr:Rachel_Stevens dbo:birthYear ?uri }",
111 "27": "SELECT ?pIraq WHERE { dbr:Iraq dbo:populationTotal ?pIraq }",
112 "28": "SELECT DISTINCT ?uri WHERE { ?uri a dbo:Country ; dbo:officialLanguage ?language }
113 GROUP BY ?uri HAVING (COUNT(?language) > 2)",
114 "29": "SELECT DISTINCT ?n WHERE { dbr:Pulp_Fiction dbo:budget ?n }",
115 "30": "SELECT DISTINCT ?uri WHERE { ?uri rdf:type dbo:Actor . ?f rdf:type dbo:Film .
116 ?f dbo:starring ?uri . } ORDER BY DESC(COUNT(DISTINCT(?f))) OFFSET 0 LIMIT 1",
117 "31": "SELECT DISTINCT ?x WHERE { ?uri dbo:deathCause ?x . }
118 ORDER BY DESC(COUNT(DISTINCT ?uri)) OFFSET 0 LIMIT 1",
119 "32": "SELECT ?uri WHERE { ?uri dbo:publisher dbr:GMT_Games }",
120 "33": "SELECT (COUNT(DISTINCT ?uri) AS ?c) WHERE { ?uri a dbo:ProgrammingLanguage }",
121 "34": "SELECT DISTINCT ?uri WHERE { ?uri a dbo:Band ; dbo:genre dbr:Metalcore
122 { ?uri dbo:hometown dbr:Australia } UNION
123 { ?uri dbo:hometown ?h . ?h dbo:country dbr:Australia } }",
124 "35": "SELECT ?Author WHERE { dbr:The_Interpretation_of_Dreams dbo:author ?Author }",
125 "36": "SELECT DISTINCT ?uri WHERE { ?uri dbo:occupation dbr:Musician .
126 ?x dbo:author ?uri . ?x rdf:type dbo:Book . }
127 ORDER BY DESC(COUNT(?x)) OFFSET 0 LIMIT 1",
128 "37": "SELECT (COUNT(DISTINCT ?uri) AS ?c) WHERE
129 { dbr:Benjamin_Franklin dbo:child ?uri }",
130 "38": "SELECT DISTINCT ?uri WHERE { dbr:The_Three_Dancers dbo:author ?person .
131 ?person dbo:movement ?uri }",
132 "39": "SELECT DISTINCT ?uri WHERE { ?uri a dbo:PoliticalParty ;
133 dbo:country dbr:Greece ; dbo:ideology dbr:Pro-Europeanism }",
134 "40": "SELECT (COUNT(DISTINCT ?uri) AS ?c) WHERE { ?uri dbo:producer dbr:Hal_Roach }",
135 "41": "SELECT DISTINCT ?uri WHERE { ?uri a dbo:Company { ?uri dbo:location dbr:Munich }
136 UNION { ?uri dbo:headquarter dbr:Munich } UNION { ?uri dbo:locationCity dbr:Munich } }",
137 "42": "SELECT DISTINCT ?uri WHERE { ?uri a dbo:SoccerClub ; dbo:league dbr:Bundesliga }",
138 "43": "SELECT DISTINCT ?runtime WHERE { dbr:Toy_Story dbo:runtime ?runtime }",
139 "44": "SELECT (COUNT(DISTINCT ?uri) AS ?c) WHERE
140 { ?uri <http://dbpedia.org/property/programme> dbr:Soyuz_programme }",
141 "45": "SELECT DISTINCT ?uri WHERE { ?uri rdf:type dbo:Holiday }",
142 "46": "SELECT ?Mexico_City WHERE { dbr:Mexico_City dbo:populationTotal ?Mexico_City }",
143 "47": "SELECT (COUNT(DISTINCT ?uri) AS ?c) WHERE { ?uri a dbo:Airline ;
144 dbo:alliance dbr:Star_Alliance }",
145 "48": "SELECT DISTINCT ?population WHERE { dbr:Eurasia dbo:populationTotal ?population }",
146 "49": "SELECT ?ele WHERE { dbr:D\u00fcrkesseldorf_Airport dbo:elevation ?ele } LIMIT 1",

```

```

147 "50": "SELECT DISTINCT ?uri WHERE { ?uri rdf:type dbo:Musical ;
148 dbo:musicBy dbr:Elton_John }",
149 "51": "SELECT DISTINCT ?uri WHERE { ?uri a dbo:Film ; dbo:starring dbr:Julia_Roberts ;
150 dbo:director dbr:Garry_Marshall }",
151 "52": "SELECT (COUNT(DISTINCT ?x) AS ?c) WHERE { dbr:Turkmenistan dbo:language ?x }",
152 "53": "SELECT DISTINCT ?uri WHERE { ?uri a dbo:Library ;
153 <http://dbpedia.org/property/established> ?year FILTER (?year < 1400) }",
154 "54": "SELECT DISTINCT ?uri WHERE { ?uri dbo:product
155 <http://dbpedia.org/resource/Slack_(software)> }",
156 "55": "SELECT DISTINCT ?uri WHERE { ?uri rdf:type dbo:LaunchPad ; dbo:operator dbr:NASA }",
157 "56": "SELECT (COUNT(?uri) AS ?count) WHERE { ?uri dbo:type dbr:States_of_Mexico }",
158 "57": "SELECT DISTINCT ?uri WHERE { ?uri a dbo:Brewery { ?uri dbo:location dbr:Australia }
159 UNION { ?uri dbo:location ?x . ?x dbo:country dbr:Australia } }",
160 "58": "SELECT (COUNT(DISTINCT ?uri) AS ?c) WHERE { ?uri a dbo:Organisation
161 { ?uri dbo:industry dbr:Telecommunication } UNION
162 { ?uri <http://dbpedia.org/property/industry> ?industry
163 FILTER regex(?industry, \"Telecommunications\") } { ?uri dbo:location dbr:Belgium }
164 UNION { ?uri <http://dbpedia.org/property/locationCountry> \"Belgium\"@en } }",
165 "59": "SELECT (COUNT(?uri) AS ?count) WHERE { dbr:Albert_Einstein dbo:knownFor ?uri }",
166 "60": "SELECT DISTINCT ?n WHERE { dbr:Brazil_national_football_team
167 <http://dbpedia.org/property/fifaMin> ?n }",
168 "61": "SELECT DISTINCT ?uri WHERE { ?uri a dbo:ProgrammingLanguage ;
169 dbo:influenced dbr:JavaScript }",
170 "62": "SELECT DISTINCT ?uri WHERE { ?uri rdf:type dbo:Song .
171 ?uri dbo:artist dbr:Bruce_Springsteen . ?uri dbo:releaseDate ?date .
172 FILTER (?date >= '1980-01-01'^^xsd:date && ?date <= '1990-12-31'^^xsd:date) }",
173 "63": "SELECT (COUNT(DISTINCT ?uri) AS ?c) WHERE { ?uri a dbo:Airline }",
174 "64": "SELECT DISTINCT ?uri WHERE { ?uri a dbo:Film ; dbo:producer dbr:Hal_Roach }"
175 },
176 "wikidata_query": {
177 "0": "SELECT DISTINCT ?uri WHERE { ?uri wdt:P161 wd:Q37079. }",
178 "1": "SELECT DISTINCT ?s1 WHERE { ?s1 wdt:P106 wd:Q49757 .
179 ?s2 wdt:P50 ?s1 . ?s2 wdt:P31 wd:Q571 . }
180 GROUP BY ?s1 ORDER BY DESC(COUNT(?s2)) OFFSET 0 LIMIT 1",
181 "2": "SELECT DISTINCT ?uri WHERE { ?uri wdt:P38 wd:Q861690 . }",
182 "3": "SELECT DISTINCT ?number WHERE { wd:Q408 wdt:P36 ?uri .
183 ?uri wdt:P1082 ?number . }",
184 "4": "SELECT DISTINCT ?val WHERE { wd:Q1198 wdt:P2046 ?val . }",
185 "5": "SELECT DISTINCT ?val WHERE { wd:Q949779 wdt:P1082 ?val . }",
186 "6": "SELECT DISTINCT ?uri WHERE { ?uri wdt:P31 wd:Q11410 ; wdt:P123 wd:Q928726 . }",
187 "7": "SELECT DISTINCT ?val WHERE { wd:Q2608065 wdt:P2130 ?val }",
188 "8": "SELECT ?uri WHERE { ?type wdt:P279* wd:Q15416 .
189 ?uri wdt:P161 wd:Q485310 . ?uri wdt:P31 ?type . }",
190 "9": "SELECT DISTINCT ?uri WHERE { wd:Q6386177 wdt:P17 ?uri . }",
191 "10": "SELECT DISTINCT ?val WHERE { wd:Q41421 wdt:P2048 ?val . }",
192 "11": "SELECT DISTINCT ?uri WHERE { wd:Q8038 wdt:P277 ?uri . }",
193 "12": "SELECT ?val WHERE { wd:Q43874 wdt:P1971 ?val . }",
194 "13": "SELECT ?o1 WHERE { wd:Q148 wdt:P38 ?o1 . } ",
195 "14": " SELECT (COUNT(?s1) as ?c) WHERE
196 { ?s1 ?p1 wd:Q739 . ?s1 wdt:P31 wd:Q34770 . }",

```

```

197 "15": "SELECT DISTINCT ?uri WHERE { ?type wdt:P279* wd:Q11424 .
198 ?uri wdt:P31 ?type ; wdt:P495 wd:Q35 . }",
199 "16": "SELECT DISTINCT ?uri WHERE { ?uri wdt:P38 wd:Q208008 . }",
200 "17": "SELECT (COUNT(DISTINCT ?uri) as ?c) WHERE { wd:Q41142 wdt:P26 ?uri . }",
201 "18": "SELECT DISTINCT (COUNT(DISTINCT ?uri) AS ?c) WHERE
202 { ?uri wdt:P106 wd:Q82955 ; wdt:P69 wd:Q49088 . }",
203 "19": "SELECT DISTINCT ?uri WHERE { ?uri wdt:P50 wd:Q23434 . ?uri wdt:P136 wd:Q4184 . }",
204 "20": "SELECT DISTINCT ?uri WHERE { ?uri wdt:P31 wd:Q476028 . ?uri wdt:P17 wd:Q29 . }",
205 "21": "SELECT DISTINCT ?uri WHERE { wd:Q458 wdt:P1082 ?uri . }",
206 "22": "SELECT ?employees WHERE { wd:Q37156 wdt:P1128 ?employees . }",
207 "23": "SELECT DISTINCT ?uri WHERE { ?uri wdt:P31 wd:Q6256 . ?uri wdt:P37 ?language . }
208 GROUP BY ?uri ORDER BY DESC(COUNT(?language)) LIMIT 1",
209 "24": "SELECT DISTINCT ?uri WHERE { ?uri wdt:P279* wd:Q38280 FILTER(?uri != wd:Q38280) }",
210 "25": "SELECT DISTINCT ?uri WHERE { wd:Q1564644 wdt:P527 ?uri }",
211 "26": "SELECT DISTINCT ?year WHERE { wd:Q241665 wdt:P569 ?date .
212 BIND(YEAR(?date) as ?year) }",
213 "27": "SELECT DISTINCT ?val WHERE { wd:Q796 wdt:P1082 ?val . }",
214 "28": "SELECT DISTINCT ?uri WHERE { ?uri wdt:P31 wd:Q6256 . ?uri wdt:P37 ?language . }
215 GROUP BY ?uri HAVING(COUNT(?language)>2)",
216 "29": "SELECT DISTINCT ?value WHERE { wd:Q104123 wdt:P2130 ?value . }",
217 "30": "SELECT DISTINCT ?uri WHERE { ?uri wdt:P106 wd:Q33999 . ?film wdt:P161 ?uri . }
218 GROUP BY ?uri ORDER BY DESC(COUNT(DISTINCT(?film))) LIMIT 1",
219 "31": "SELECT ?uri WHERE { ?s wdt:P509 ?uri . } GROUP BY ?uri
220 ORDER BY DESC(COUNT(DISTINCT ?s)) LIMIT 1",
221 "32": "SELECT ?uri WHERE { ?uri wdt:P31 wd:Q131436 . }",
222 "33": "SELECT (COUNT(DISTINCT ?sub) AS ?count) WHERE { ?sub wdt:P31 wd:Q9143 . }",
223 "34": "SELECT DISTINCT ?uri WHERE { ?uri wdt:P31 wd:Q215380 ;
224 wdt:P495 wd:Q408 ; wdt:P136 wd:Q183862 . }",
225 "35": "SELECT DISTINCT ?uri WHERE { wd:Q726382 wdt:P50 ?uri . }",
226 "36": "SELECT DISTINCT ?uri WHERE { ?uri wdt:P106 wd:Q639669 . ?x wdt:P31 wd:Q571 ;
227 wdt:P50 ?uri . } GROUP BY ?uri ORDER BY DESC(COUNT(?x)) LIMIT 1",
228 "37": "SELECT (COUNT(DISTINCT ?uri) as ?c) WHERE { wd:Q34969 wdt:P40 ?uri . }",
229 "38": "SELECT DISTINCT ?uri WHERE { wd:Q3235867 wdt:P170 ?painter .
230 ?painter wdt:P135 ?uri . }",
231 "39": "SELECT DISTINCT ?uri WHERE { ?uri wdt:P31 wd:Q7278 ;
232 wdt:P17 wd:Q41 ; wdt:P1142 wd:Q3781399 . }",
233 "40": "SELECT (COUNT(?uri) as ?c) WHERE { ?uri wdt:P162 wd:Q72792 . }",
234 "41": "SELECT DISTINCT ?uri WHERE { ?type wdt:P279* wd:Q4830453 .
235 ?uri wdt:P31 ?type ; wdt:P159 wd:Q1726 . }",
236 "42": "SELECT DISTINCT ?uri WHERE { ?uri wdt:P31 wd:Q476028 ; wdt:P118 wd:Q82595 . }",
237 "43": "SELECT ?val WHERE { wd:Q171048 wdt:P2047 ?val . }",
238 "44": "SELECT (COUNT(DISTINCT ?uri) AS ?c) WHERE { ?uri wdt:P361 wd:Q207329 }",
239 "45": "SELECT ?uri WHERE { ?uri wdt:P31 wd:Q1445650 . }",
240 "46": "SELECT DISTINCT ?uri WHERE { wd:Q1489 wdt:P1082 ?uri . }",
241 "47": "SELECT (COUNT(DISTINCT ?uri) as ?c) WHERE { ?uri wdt:P114 wd:Q189709 . }",
242 "48": "SELECT DISTINCT ?val WHERE { wd:Q5401 wdt:P1082 ?val . }",
243 "49": "SELECT DISTINCT ?uri WHERE { wd:Q58226 wdt:P2044 ?uri . }",
244 "50": "SELECT DISTINCT ?uri WHERE { ?uri wdt:P31 wd:Q58483083 ; wdt:P86 wd:Q2808 . }",
245 "51": "SELECT DISTINCT ?uri WHERE { ?uri wdt:P31 wd:Q11424 .
246 ?uri wdt:P57 wd:Q315087 . ?uri wdt:P161 wd:Q40523 . } ",

```

```

247 "52": "SELECT (COUNT(DISTINCT ?uri) as ?c) WHERE { wd:Q874 wdt:P2936 ?uri . }",
248 "53": "SELECT DISTINCT ?uri WHERE { ?uri wdt:P31 wd:Q7075 ;
249 wdt:P571 ?date . FILTER (YEAR(?date) < 1400) }",
250 "54": "SELECT DISTINCT ?uri WHERE { ?uri wdt:P1056 wd:Q17130715}",
251 "55": "SELECT DISTINCT ?uri WHERE { ?uri wdt:P31 wd:Q194188 ; wdt:P137 wd:Q23548 . }",
252 "56": "SELECT (COUNT(?uri) AS ?count) WHERE { ?uri wdt:P31 wd:Q15149663 . }",
253 "57": "SELECT DISTINCT ?uri WHERE { ?uri wdt:P31 wd:Q131734 . ?uri wdt:P17 wd:Q408 . }",
254 "58": "SELECT DISTINCT ?uri WHERE { { ?uri wdt:P31 wd:Q43229 }
255 UNION { ?uri wdt:P31/(wdt:P279*) wd:Q43229 } .
256 ?uri wdt:P452 wd:Q418 . ?uri wdt:P17 wd:Q31 . }",
257 "59": "SELECT (COUNT(DISTINCT ?uri) as ?cnt) WHERE { wd:Q937 wdt:P800 ?uri . }",
258 "60": "SELECT ?value WHERE { wd:Q83459 p:P1352 ?statement .
259 ?statement pq:P459 wd:Q180825 ; ps:P1352 ?value . }
260 ORDER BY DESC(?value) LIMIT 1",
261 "61": "SELECT DISTINCT ?uri WHERE { wd:Q2005 wdt:P737 ?uri . }",
262 "62": "SELECT DISTINCT ?uri WHERE { ?uri wdt:P31 wd:Q7366 ; wdt:P577 ?date ;
263 wdt:P175 wd:Q1225 . BIND(YEAR(?date) as ?year) .
264 FILTER(?year >= 1980 && ?year <= 1990) }",
265 "63": "SELECT (COUNT(DISTINCT ?uri) AS ?uri) WHERE { ?uri wdt:P31 wd:Q46970 }",
266 "64": "SELECT DISTINCT ?uri WHERE { ?uri wdt:P31 wd:Q11424 . ?uri wdt:P162 wd:Q72792 . }"
267 }
268 }

```

**Code 2: JSON for QALD Test Dataset**

### APPENDIX 3: TASK DISTRIBUTION

As the project focuses on the research and development of the GraphRAG, the task was separated into two: research and development phases. To be specific, most of the research phase was conducted by Jaycent Gunawan Onggris, whereas most of the development phase was conducted by Eduardus Tjitrahardja.

1. **Chapter 1:** Jaycent organized the research overview and structured the outline of the paper. Together, Jaycent and Eduardus collaboratively defined the research questions, objectives, scope, and stages.
2. **Chapter 2:** Jaycent conducted the literature review on KGs, several examples of LLMs, and GraphRAG. Eduardus focused on literature regarding LLMs (including Transformers) and RAG in general.
3. **Chapter 3:** Jaycent contributed by detailing the research methodology, experimental setup, evaluation metrics, and information on knowledge bases and ontologies. He also transformed the local KG to RDF and created the local KG dataset generator. Meanwhile, Eduardus focused on pre-processing the Wikidata and DBpedia (QALD) datasets.
4. **Chapter 4:** The original pipeline (v1) was collaboratively designed by Jaycent and Eduardus, while the improved pipeline (v2) was primarily conceptualized by Jaycent, with Eduardus refining it during implementation. The system implementation was conducted entirely by Eduardus.
5. **Chapter 5:** Eduardus led the analysis of the improved pipeline and conducted the ablation study to identify key components of the system. Jaycent analyzed the results of the original pipeline and contributed to the broader discussion of LLMs and knowledge bases. Dataset issues were initially identified by Eduardus and later elaborated in the paper by Jaycent.
6. **Chapter 6:** Jaycent constructed the conclusion subsection, while the future work section was co-designed by both Jaycent and Eduardus.

## APPENDIX 4: FROG GITHUB README

### Abstract

The rise of large language models (LLMs) has advanced information retrieval, but issues like limited knowledge updating, lack of openness, and hallucinations persist. Retrieval-Augmented Generation (RAG) addresses these, though it lacks interpretability due to reliance on vector-based representations. Our research presents a RAG framework using a knowledge graph (KG) as the primary knowledge base, with only open-source components to allow for user customization. The main pipeline consists of entity linking, retrieval using verbalized sentences or SPARQL query generation, and answer generation, integrated with ontology (properties and classes) retrieval via a vector database. The pipeline is tested on Wikidata, DBpedia, and a local domain-specific KG, achieving accuracies of 0.458, 0.517, and 0.805, respectively. An ablation study further reveals that ontology retrieval is the most critical component in providing context to the LLM for generating accurate SPARQL queries.

### Installation

1. Ensure you have PyTorch installed. (Recommended to have PyTorch installed with GPU support if you want to run a local LLM.)

2. Install dependencies:

```
1 pip install -r requirements.txt --no-dependencies
2
```

3. Set up the Weaviate vector database:

```
1 docker compose up -d
2
```

Alternatively, you can use Weaviates cloud service.

4. Create a .env file and fill in the following variables:

```
If you are using Hugging Face API for models
HF_TOKEN=

If you are using Weaviate cloud service
WEAVIATE_URL=
```

WEAVIATE\_API\_KEY=

## Example Usage

```
1 from rag import WikidataGraphRAG
2
3 rag_engine = WikidataGraphRAG(
4 model_name="Qwen/Qwen2.5-7B-Instruct",
5 max_new_tokens=1500,
6 use_local_model=True,
7 use_local_weaviate_client=True,
8)
9
10 response = rag_engine.chat(
11 "How many programming languages are there?",
12 verbose=1, use_cot=True, try_threshold=5
13)
14 print(response)
```

## Evaluation

### Performance

#### Wikidata

Configuration	Jaccard Similarity
Mistral NeMo 12B	0.423
LLaMA 3.1 8B	0.427
Qwen2.5 Coder 7B	0.428
<b>Qwen2.5 7B</b>	<b>0.458</b>

**Table 1:** Experiment Results on the Downsampled Wikidata QALD-9-Plus Dataset

#### DBpedia

Configuration	Jaccard Similarity
Mistral NeMo 12B	0.450
LLaMA 3.1 8B	0.444
Qwen2.5 Coder 7B	0.442
<b>Qwen2.5 7B</b>	<b>0.517</b>

**Table 2:** Experiment Results on the Downsampled DBpedia QALD-9-Plus Dataset

### Local KGs

Configuration	Jaccard Similarity
Mistral NeMo 12B	0.805
LLaMA 3.1 8B	0.778
Qwen2.5 Coder 7B	0.778
Qwen2.5 7B	0.805
<b>Mistral NeMo 12B w/o Verbalization</b>	<b>0.949</b>
<b>Qwen2.5 7B w/o Verbalization</b>	<b>0.949</b>

**Table 3:** Experiment Results on the Local Curriculum KG Dataset

## Run Your Own

You can evaluate the framework using the provided `eval.py` script. Below is an overview of the evaluation parameters and their usage.

## Command-Line Arguments

- `--knowledge-source`: Specify the knowledge source to use. Options: `wikidata`, `dbpedia`, `enterprise`.
- `--model-name`: The name of the model to use.
- `--local-model`: Use the model locally (default: `True`).
- `--local-weaviate-client`: Use the Weaviate client locally (default: `True`).
- `--max-new-tokens`: Maximum number of new tokens (default: `1500`).
- `--test-df-path`: Path to the test dataframe (default: `data/eval/qald-9-downsampled-test-latest.json`).
- `--always-use-generate-sparql`: Always use SPARQL query generation (default: `False`).
- `--use-cot`: Use Chain of Thought reasoning (default: `True`).
- `--llm-try-threshold`: LLM try threshold for retrying queries (default: `10`).
- `--log-file-path`: Path to the log file (default: `logs/eval_log.txt`).
- `--use-transform-factoid`: Enable transform factoid reasoning (default: `False`).

## Example Command

```
1 python eval.py \
2 --knowledge-source="wikidata" \
3 --model-name="Qwen/Qwen2.5-7B-Instruct" \
4 --log-file-path="logs/log.txt" \
5 --test-df-path="data/eval/qald-9-downsampled-test-latest.json" \
6 --llm-try-threshold=5 \
7 --local-model \
```

```
8 --no-local-weaviate-client \
9 --use-cot
```

## Customization

### Enterprise Knowledge Graph Integration

To integrate a custom enterprise knowledge graph:

1. Specify the `turtle_file_path` parameter when initializing `EnterpriseGraphRAG`.
2. Modify the prompt templates in `rag/enterprise.py` and `ENTERPRISE_GENERATE_SPARQL_FEW_SHOTS` few shots in `few_shots.py` to suit your use case.

### Acknowledgements

This framework is built with open-source tools to foster transparency and customization in the research community. Contributions and suggestions are welcome! We also extend our gratitude to Wikimedia Foundation Indonesia for their generous support in funding this research project.