

The Australian National University
2600 ACT | Canberra | Australia



Australian
National
University

School of Computing
College of Engineering, Computing
and Cybernetics (CECC)

Local Holistic Knowledge Graph Construction From Academic Papers Using Large Language Models Through In-Context Learning

— Honours project (S1 2024)

A thesis submitted for the degree
Bachelor of Advanced Computing

By:
Haoting Chen

Supervisors:
Dr. Sergio Rodríguez Méndez
Dr. Pouya Ghiasnezhad Omran

November 2024

Declaration:

I declare that this work:

- upholds the principles of academic integrity, as defined in the [Academic Integrity Rule](#);
- is original, except where collaboration (for example group work) has been authorised in writing by the course convener in the class summary and/or LMS course site;
- is produced for the purposes of this assessment task and has not been submitted for assessment in any other context, except where authorised in writing by the course convener;
- gives appropriate acknowledgement of the ideas, scholarship and intellectual property of others insofar as these have been used;
- in no part involves copying, cheating, collusion, fabrication, plagiarism or recycling.

I acknowledge that I am expected to have undertaken Academic Integrity training through the Epigeum Academic Integrity modules prior to submitting an assessment, and so acknowledge that ignorance of the rules around academic integrity cannot be an excuse for any breach.

November (2024), Haoting Chen

Acknowledgements

I would like to express my sincere gratitude to my supervisors, Dr. Sergio Rodriguez Mendez and Dr. Pouya Ghiasnezhad Omran, who are willing to spend both their working time and personal time to help me address the challenges in my research. Without their support, my research would not have progressed this far.

Abstract

This research presents a novel Knowledge Graph Construction (KGC) pipeline, LLM-KGC-H, which is capable of converting semi-structured text documents, primarily academic papers, into their Knowledge Graphs (KG) representation that captures the relationship between entities inside the text in a structured way, such as (“ANU”, “located in”, “Canberra”), where “ANU” and “Canberra” are nodes and “located in” is an edge in the graph. The pipeline is constructed by leveraging Large Language Models (LLMs), primarily LLM decoders, to achieve various Natural Language Processing (NLP) tasks required for KGC using different input prompts with instruction and only zero or several examples, also referred to as in-context learning. This is as opposed to the traditional way of KGC that depends mostly on fine-tuning LLMs, mainly LLM encoders, on some KGC dataset-related datasets, which may require significant computational resources during training and most importantly, the finalised model may not scale well on unseen data. The output KG is considered local and holistic, which means one output KG focuses on capturing the relationship presented in one document only, but from the view of the whole document, it aims to capture more entities and their relationship that traditional KGC cannot. The research process involved undertaking an extensive literature review of recent papers on the overall design principle of KGC pipelines or a single KGC subtask, implementing and experimenting with the techniques gathered, incorporating new ideas, and evaluating and presenting a KGC pipeline artifact. This research holds significance in expanding the ANU Academic Scholarly Knowledge Graph (ASKG) database, which mainly consists of semi-structured papers with metadata and hierarchical relations between sections, paragraphs, and sentences rather than deep semantic relations between entities. Our evaluation shows that while using LLM decoders enables our model to capture most relations between entities in papers and some relations that traditional KGC cannot, the autoregressive nature of LLM decoders can also significantly impact the pipeline’s efficiency, especially for long documents. The hallucination issue is another major challenge hindering the pipeline’s ability to generate high-precision KGs.

Table of Contents

1	Introduction	1
1.1	Knowledge Graph (KG)	1
1.2	Knowledge Graph Construction (KGC)	2
1.3	KGC Using Large Language Models (LLMs)	3
1.4	ANU Academic Scholarly Knowledge Graph (ASKG)	6
1.5	Summary and Research Goals	7
1.6	Thesis Outline	7
2	Background	9
2.1	Various KGs	9
2.2	Some Properties of RDF-KG	11
2.2.1	Identity	11
2.2.2	Storage	12
2.2.3	Flexibility	12
2.3	Definition of Entity, RDF-Graph, RDF-KG	14
2.3.1	Entity and Literal	15
2.3.2	RDF-Graph	15
2.3.3	RDF-KG	16
2.4	LLMs	17
2.4.1	Encoders: Training, Inference, and Complexity	17
2.4.2	Decoders: Training, Inference, and Complexity	18
2.4.3	Encoders and Fine-Tuning	20
2.4.4	Decoders and In-Context Learning	20
2.4.5	Complexity Issue in Decoder	21
2.4.6	Hallucination Issues in Decoder	21
3	Literature Review	23
3.1	Traditional KGC Approach	23
3.1.1	Schema Definition	24
3.1.2	Named Entity Recognition and Entity Typing	24
3.1.3	Coreference Resolution and Entity Disambiguation	25
3.1.4	Relation Extraction	26

Table of Contents

3.2	LLM-Based Implementation of Traditional KGC	26
3.2.1	Named Entity Recognition and Entity Typing	26
3.2.2	CorefERENCE Resolution, Entity Disambiguation, and Relation Extraction	30
3.2.3	Literature Based on Traditional KGC	30
3.3	Reflection on Traditional KGC	33
3.3.1	Training Involved	33
3.3.2	Predefined Types and Predicates	34
3.3.3	Multi-Stage KGC	34
3.3.4	Relations Between Named Entities Only	34
3.3.5	Remarks	35
3.4	Related Work on Addressing Traditional KGC	35
3.4.1	Examples	35
3.4.2	Remarks	36
4	Design of the Local Holistic KGC Approach	37
4.1	Demonstration of the Local Holistic KGC Approach	37
4.1.1	Entity Extraction, Typing, and Classification	40
4.1.2	Entity Disambiguation and Coreference Resolution	41
4.1.3	Relation Extraction	42
4.1.4	Schema Generation	42
4.2	Comparison to Traditional KGC	42
4.3	Comparison to other Non-Traditional KGC Approach	42
4.4	SWOT Analysis for Our Holistic KGC Approach	44
4.4.1	Strengths	44
4.4.2	Weaknesses	44
4.4.3	Opportunities	45
4.4.4	Threats	45
5	Overview of the Implementation of the Local Holistic KGC Pipeline	47
5.1	Interface	47
5.1.1	LLM Encoders and Decoders Used	48
5.1.2	Development Set Used	49
5.1.3	High-Level Input and Output Data Structure	49
5.1.4	Low-Level Input (TTL and JSON)	50
5.1.5	Low-Level Output (TTL and JSON)	52
5.2	Pipeline Structure	54
5.2.1	Stage 0: Data Pre-Processing	55
5.2.2	Stage 1: Entity Extraction, Typing, and Classification	55
5.2.3	Stage 2: Entity Disambiguation and Coreference Resolution	58
5.2.4	Stage 3: Local Relation Extraction	60
5.2.5	Stage 4: Global Relation Extraction	61
5.2.6	Stage 5: Schema Generation	62
5.2.7	Stage 6: KG Post-Processing	62

Table of Contents

5.3	Execution Diagram	62
5.4	Decision Made and Analysis	65
5.4.1	Single Paper as the Primary Development Set	65
5.4.2	Modular Design	65
5.4.3	Using JSON instead of TTL	66
6	Stage I: Entity Mention Extraction, Classification, and Typing	69
6.1	Algorithm	69
6.2	Prompts	74
6.3	Assumptions Made	76
6.3.1	A1: LLMs are Knowledgeable	76
6.3.2	A2: All Sections Fit the Context Limit of the LLM Used	76
6.3.3	A3: Identical Mentions in a Local Text Refer to the Same Thing	76
6.3.4	A4: An Entity Mention Extracted at the Paragraph or Section Level Means All Sentence Containing the Term Same As the Entity Mention Have the Entity Mention	77
6.3.5	A5: All Named Entities and Concepts Extracted Are Indeed Named Entities and Concepts	78
6.4	Limitations	78
6.4.1	L1: Only Mentions That Syntactically Appear in Text Are Extracted	78
6.4.2	L2: Only Entities That Have at least One Mentions Are Captured	78
6.5	Complexity Analysis	79
6.5.1	Analysis Approach	79
6.5.2	Stage 1 Complexity	79
6.6	Decisions Made and Analysis	79
6.6.1	Simple Prompts vs. Detailed Prompts	79
6.6.2	Hierarchical Extraction	80
6.6.3	Explicit Entity Classification vs. Classification via Set Subtraction	82
6.6.4	JSON Output vs. Customised-Format Output	84
7	Stage II: Entity Disambiguation and Coreference Resolution	85
7.1	Algorithm	85
7.2	Prompts	91
7.3	Assumptions Made	92
7.4	Limitations	93
7.5	Complexity Analysis	93
7.6	Decisions Made and Analysis	94
7.6.1	Using Description for Embedding Generation	94
7.6.2	Disjoint Maximal Cliques vs. Disjoint Connected Subgraphs	95
8	Stage III: Local Relation Extraction	97
8.1	Algorithm	97
8.2	Prompts	101
8.3	Assumptions Made	101

Table of Contents

8.4 Limitations	102
8.5 Complexity Analysis	102
8.6 Decisions Made and Analysis	102
9 Stage IV: Global Relation Extraction	105
9.1 Algorithm	105
9.2 Prompts	108
9.3 Assumptions, Decision Made, and Limitations	109
9.4 Complexity Analysis	110
10 Stage V: Schema Generation	111
10.1 Algorithm	111
10.2 Prompts	116
10.3 Assumptions Made	117
10.4 Limitations	117
10.5 Complexity Analysis	118
10.6 Design Decision	118
11 Evaluation	121
11.1 Experiment Setup	121
11.1.1 Dataset 1: Five Full Papers	121
11.1.2 Dataset 2: SciERC	122
11.1.3 Platform	122
11.1.4 LLMs	122
11.2 General Evaluation	123
11.2.1 Discussion on the KGs	131
11.2.2 Discussion on the Complexity	131
11.3 Evaluation via Reverse Engineering	132
11.3.1 Evaluation Approach	134
11.3.2 Evaluation Results	137
11.3.3 Remarks	137
11.4 Evaluation via Application	138
11.4.1 Evaluation Approach	138
11.4.2 Evaluation Results	142
11.4.3 Remarks	144
11.5 Manual Evaluation	145
11.6 Ablation Study	151
11.7 Summary of Limitations	156
12 Concluding Remarks	157
12.1 Research Summary	157
12.2 Conclusion	158
12.3 Future Work	159
12.3.1 “Scaling Up”	159

Table of Contents

12.3.2 “Scaling Out”	160
12.3.3 Combining of KGs and LLMs	161
12.4 Ethical Considerations	161
12.4.1 Privacy Risks in KGC	161
12.4.2 Environmental Impact of KGC	162
12.4.3 Rapid Advancement of AIs	162
Bibliography	165

Chapter 1

Introduction

1.1 Knowledge Graph

A Knowledge Graph (KG), in general, refers to a broad type of graph-based data model capable of storing and representing knowledge (Hogan et al., 2020). Like graphs in mathematics, a KG consists of nodes and edges, where nodes can represent entities or literals and edges represent predicates (Hogan et al., 2020). An entity is any *distinct* and *identifiable* thing in the real world that includes but is not limited to named entities (e.g., ANU) and entity types or concepts (e.g., University) (Hogan et al., 2020; Oxford University Press, 2024). A literal, on the other hand, is a value often used for describing an entity and is not necessarily distinct (Hogan et al., 2020) (e.g., one of the names of the entity ANU). When two nodes in a KG are connected by an edge, a single piece of knowledge is formed, e.g., (ANU, is a, University). Therefore, a KG with nodes connected by edges can then represent a collection of knowledge. It is worth noting that apart from its graphical form, the KG shown in Figure 1.1 can also be treated as a *set* of (subject, predicate, object) triples telling the relationship between entities or entities and literals.

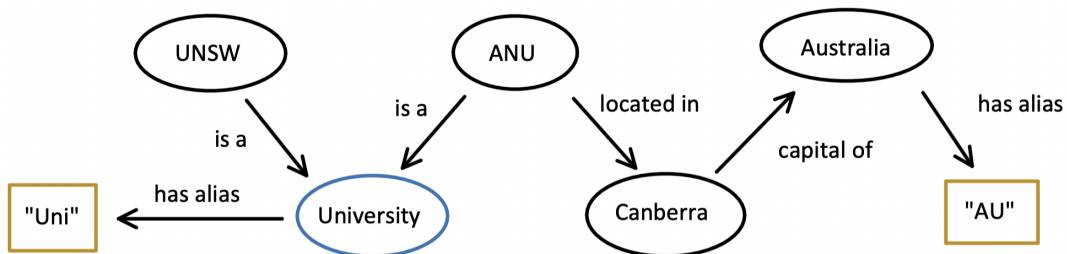


Figure 1.1: An intuitive view of a KG

1 Introduction

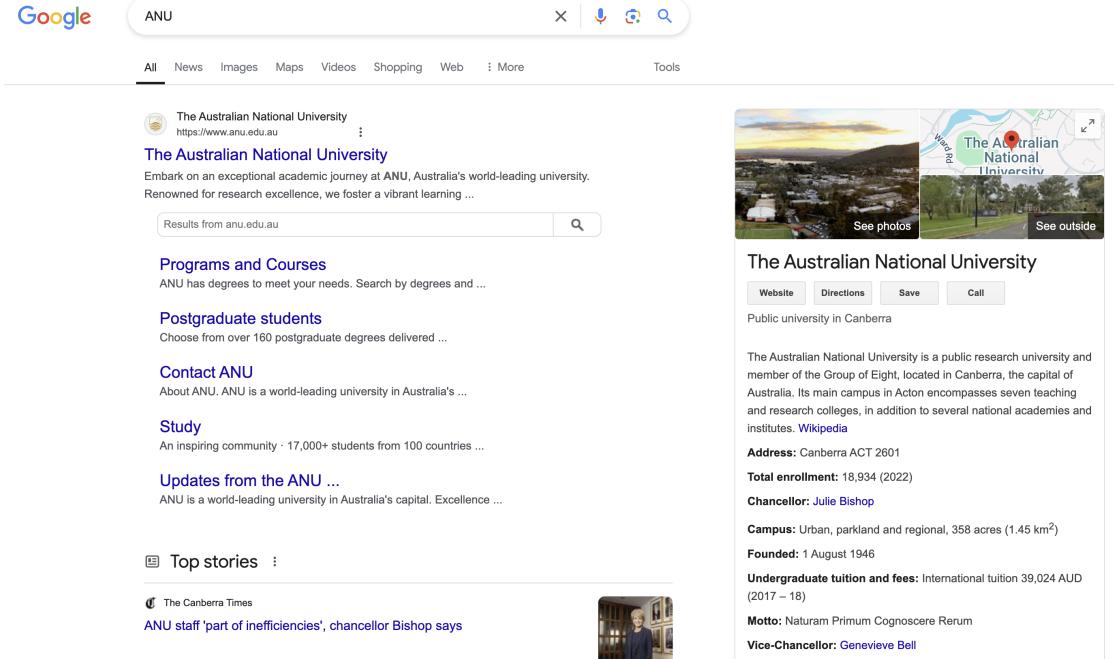


Figure 1.2: An application of KGs: Google Knowledge Panels (right)

Compared to the raw text like "ANU is a university located in Canberra", the data represented in KGs is *structured* (Hogan et al., 2020; Pan et al., 2023), thereby more machine-readable. For example, using some simple logic is sufficient to visit all the neighbours of the ANU node and, hence, retrieve lots of information about ANU. This makes KGs useful in many information retrieval tasks, such as web searches (Pan et al., 2023), if the sources are represented as KGs. In fact, as Figure 1.2 shows, the Google Knowledge Panel on the right of the result page in a Google search is powered by Google Knowledge Graph (Singhal, 2012), which allows the information to be retrieved quickly, concisely, and thoroughly. Google is also considered the one who first introduced and popularised the modern KG in 2012 (Hogan et al., 2020).

1.2 Knowledge Graph Construction

Despite the usefulness of KGs, the Knowledge Graph Construction (KGC) task is challenging (Hogan et al., 2020). It requires a detailed analysis of the sources, such as text documents, to extract entities and their relations for building nodes and edges (Hogan et al., 2020). In text documents especially, some entities may be represented explicitly, e.g., ANU, while others' representations are implicit, e.g., It and They. In addition, some relations can be found easily by reading a single sentence, e.g., (ANU, located in, Canberra), while others may span across the whole document, e.g., (This paper, concludes, XXX). Therefore, this may require significant human effort or machines to

1.3 KGC Using Large Language Models (LLMs)

have a strong understanding of natural language (Hogan et al., 2020; Pan et al., 2023). In our study, we focused on building KGs by machines only without any human interference, i.e., *automatic* KGC. Since it is also practically impossible to capture all human knowledge in one huge KG, our focus is to build single *local* KGs for each document that captures only the knowledge presented in the document. Even though they are local when built, it is possible to align them together to form a larger KG, hence representing more knowledge (Hogan et al., 2020). This is why building local KGs is meaningful.

The current automatic local KGC processes from text documents have several limitations. (1) First, they often focus on extracting named entities that fall within a predefined set of concepts (also known as types), e.g., `Person`, `Location`, and `Organisation` (Section 3.3.2). However, the types of named entities in a document could vary between different documents. For example, if the document aims to introduce some specific `Algorithms`, but our predefined types are `Person`, `Location`, and `Organisation` only, we can miss a lot of potential nodes in our constructed KG. (2) Similarly, many recent KGC methods can only extract relations between named entities with a predicate belonging to a predefined predicate list (Section 3.3.2), and (3) these relations are usually extracted from the sentence level only (3.4.2). (4) Moreover, only focusing on extracting relations between named entities is too limited to capture a broader range of knowledge in documents (Section 3.3.4). Many texts may present knowledge between two concepts, e.g., (`Programming Languages`, `automate`, `Repetitive Tasks`) or a named entity and a concept, e.g., (`JavaScript`, `enables`, `Web Development`), etc. (5) Even though several more recent studies have overcome the limitations of building knowledge graphs using predefined types and predefined relations, some of them only focus on a specific subtask of the KGC rather than presenting a full KGC pipeline (Section 3.4.2).

Therefore, it is necessary to design and implement a more *holistic* automated local KGC pipeline, spanning all the necessary stages of KGC. We will revisit the limitations of the current KGC pipelines and reveal the potential reasons for the limitations in the literature review section 3.

1.3 KGC Using Large Language Models

To achieve our local holistic KGC pipeline from text documents, we used Large Language Models (LLM) as our tools. There are three main types of LLMs: encoders, decoders, and encoder-decoder models (Pan et al., 2023). In this study, we will focus on encoders and decoders only.

In general, LLM decoders, like GPT (OpenAI, 2024a) and LLaMA (Meta, 2024), take texts in natural language as inputs, also known as prompts, and generate texts that continue coherently from the input texts, usually the answers (Pan et al., 2023). This is useful for doing generative tasks. The Named Entity Recognition (NER), Entity Typing (ET), and Relation Extraction (RE), which are part of the traditional KGC process, can be modelled as generative tasks (Pan et al., 2023). NER helps create the named

1 Introduction

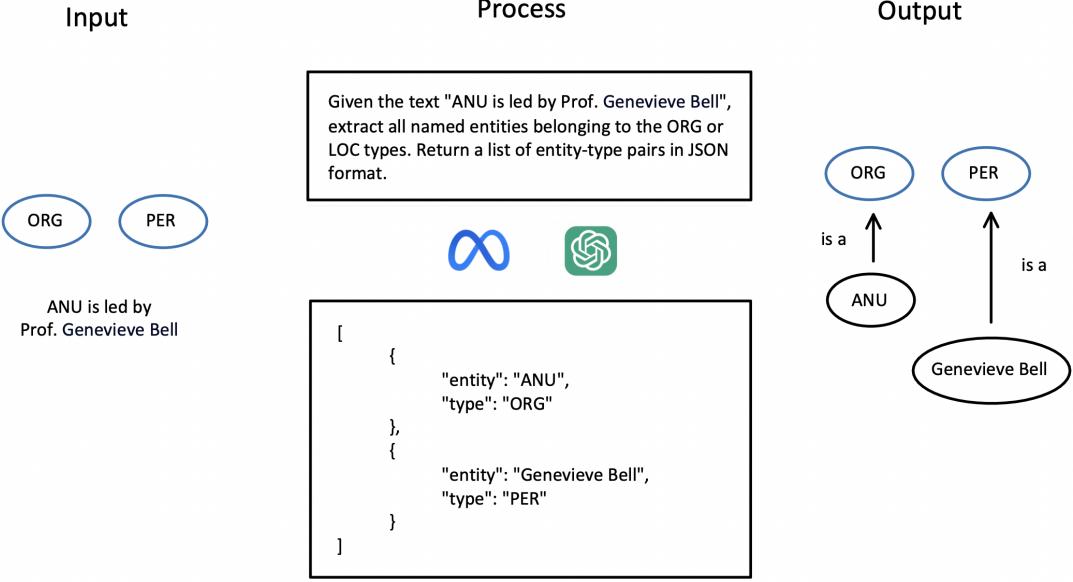


Figure 1.3: NER and ET using LLM decoders

entity nodes in the target KG based on the given text (Pan et al., 2023), ET connects the named entity nodes and the concept nodes with ontological predicates (e.g., `is a`) (Pan et al., 2023), and RE connects any entity nodes with any predicates (Pan et al., 2023). NER and ET are often performed at the same time (Xie et al., 2023; Ashok and Lipton, 2023; Wang et al., 2023). The high-level processes of using LLM decoders to perform NER, ET, and RE are shown in Figure 1.3 and Figure 1.4.

LLM encoders, such as BAAI's bge-large-en (Xiao et al., 2024), on the other hand, take texts in natural language as inputs and provide their language understanding of the original text in dense vectors, which is called embedding (Pan et al., 2023). Embeddings are generally human-unreadable but machine-readable. For example, the inputs with similar semantic meanings will have high cosine similarity when compared to their embeddings (Pan et al., 2023). This is useful for doing classification tasks. For example, Coreference Resolution (CR), which is part of the traditional KGC process, can be modelled as a classification task to decide whether two entities mentioned, appearing differently, refer to the same actual entities. The process for CR is shown in Figure 1.5 (Pan et al., 2023).

Therefore, LLMs are useful and essential for building our KGC pipeline. In fact, NER, ET, and RE can also be modelled as classification tasks handled by encoders, whereas CR can also be treated as a generation task performed by decoders. More details will be discussed in the literature review section 3.

1.3 KGC Using Large Language Models (LLMs)

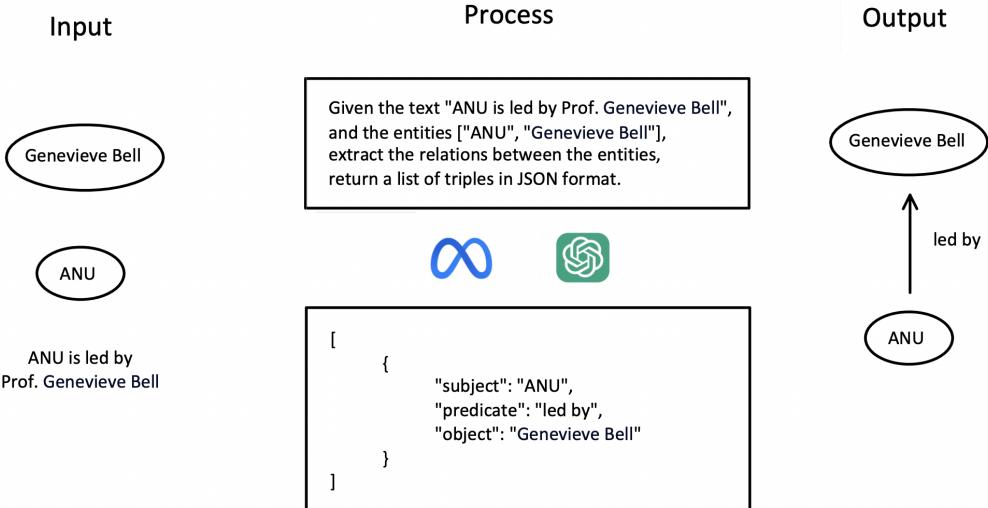


Figure 1.4: RE using LLM decoders

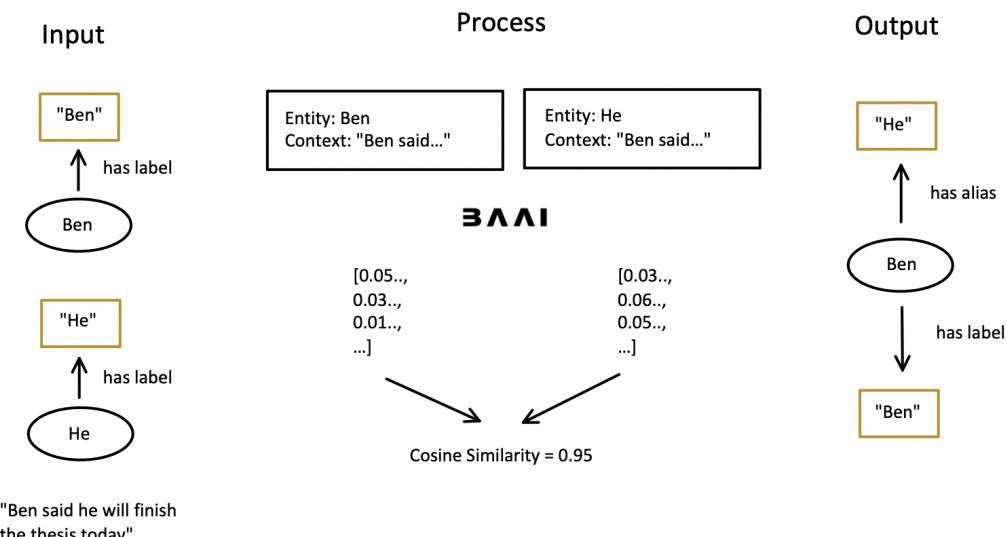


Figure 1.5: CR using LLM encoders

1 Introduction

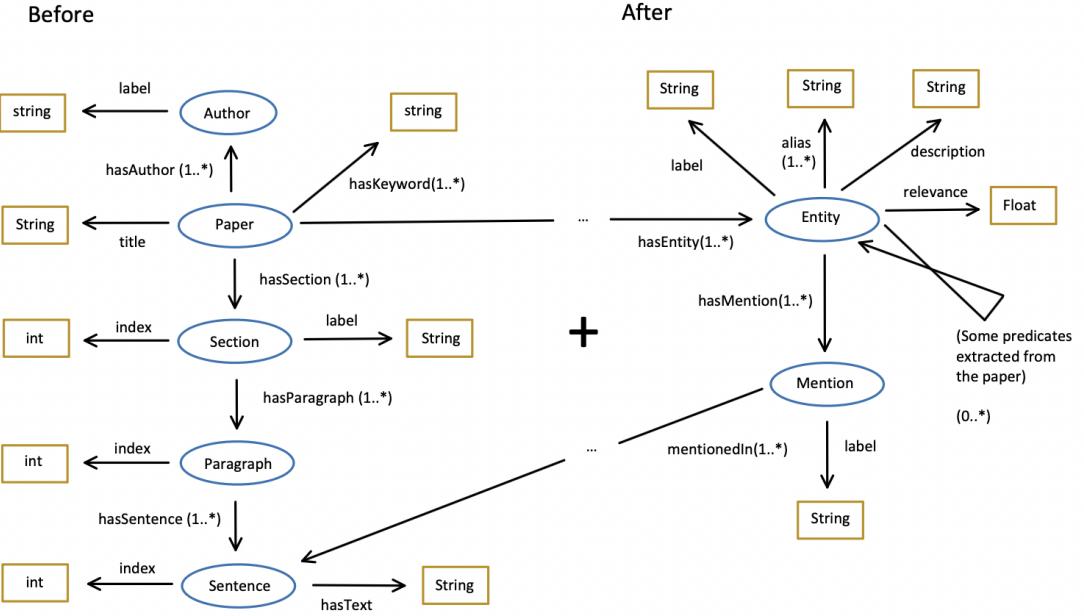


Figure 1.6: The ANU ASKG structure that this project focuses on: Before and After (Partial)

The full ontology of ASKG is available at <https://w3id.org/kgcp/DDM>.

1.4 ANU Academic Scholarly Knowledge Graph

ANU Academic Scholarly Knowledge Graph (ASKG) is a KG-based database for storing academic papers (Jia et al., 2024; Zhang et al., 2023). However, ASKG is currently mainly able to capture the knowledge of papers at a high granular level, i.e., the metadata knowledge, e.g., the relationship between papers and authors, sections and paragraphs, and sentences and text, as shown in Figure 1.6 (left)¹, whereas the knowledge about the actual content of papers is lacking, i.e., the relationship between entities presented in the document (Jia et al., 2024). For example, ASKG may contain the metadata knowledge like (**Sentence 1**, **has text**, "ANU is a university"), but does not have the content-related knowledge of (**ANU**, **is a**, **University**).

In fact, the primary motivation of this study is to find a way to expand the current ASKG to consider knowledge at the content level. However, it turns out that the current off-the-shelf KGC methods are not enough to capture more holistic knowledge about the papers. For example, current KGC methods may focus on extracting named entities belonging to a predefined set of types (Pan et al., 2023). However, documents, especially academic papers, tend to introduce new knowledge, and hence, there is a higher possibility that the named entities presented in a paper do not belong to any predefined types.

¹This structure is called **Deep Document Model** (DDM).

1.5 Summary and Research Goals

Therefore, the targeted input of our KGC pipeline is any single semi-structured academic text document from ASKG, represented as a local metadata KG, containing basic hierarchical knowledge about its section, paragraph, and sentence layouts. Our output is the original metadata KG, plus the deep semantic knowledge about the entities presented in the content, as shown in Figure 1.6.

Compared to raw text as input, the hierarchy information about each paper could be particularly useful when the pipeline is programmed to loop through the document sentence by sentence, paragraph by paragraph, to extract information to build KGs, saving a lot of preprocessing. Our pipeline does not start from raw text not only because ASKG is already semi-structured, but also because the conversion between the unstructured raw text or files of academic papers into its semi-structured representation has been widely studied by other ANU researchers ([Zhang et al., 2023](#); [Jia et al., 2024](#)), one of the outcomes is [Jia et al. \(2024\)](#)'s Deep Document Model (DDM). In fact, the current stage of part of ASKG, which is shown in Figure 1.6 (left), is exactly part of output of DDM ([Jia et al., 2024](#)). Hence, by utilising their outputs, our pipelines can eventually be connected.

Our outputs, as Figure 1.6 (right) shows, will contain a set of entities from the input paper, where each entity contains a label, i.e., the most representative name, some aliases, and a description. Each entity may be connected with zero or more other entities by some predicates extracted from the paper, e.g., (LLMs, enable, Knowledge Graph Construction). The relevance score means how relevant is the entity to the document. Each Entity node will have at least one Mention node, which stores the exact forms of an Entity being mentioned in the original text and the specific sentence location. For example, the entity LLM may appear as Large Language Model in Sentence 1, but LLM elsewhere.

1.5 Summary and Research Goals

This chapter has briefly introduced “KG”, “KGC”, “KGC using LLMs”, and “KGC from academic papers” by degrees. We have shown some of our research motivations, e.g., the limitations of the current KGC methods and the lack of content-level knowledge in ASKG. These can be further derived into two main research goals.

1. Design and implement a local holistic KGC pipeline prototype that takes semi-structured academic papers as inputs.
2. Briefly discover the effectiveness of using LLMs in local holistic KGC through the evaluation of the output KGs of the pipeline prototype.

1.6 Thesis Outline

The later chapters of this thesis is organised in the following ways:

1 Introduction

- Section 2 (Background) will provide the essential concepts about KGs and LLMs relevant to this study and define the specific subtype of KG our study focuses on.
- Section 3 (Literature Review) will focus on the general traditional KGC approach that is widely adopted by the literature and highlight its limitations. It will also provide some recent alternative solutions to the traditional KGC approach.
- Section 4 (Pipeline Design) will provide our solution to the limitations of the traditional KGC approach, i.e., the design of the holistic KGC pipeline.
- Section 5 (Pipeline Implementation Overview) will have an overview of the final pipeline artifact of this study.
- Section 6 to 10 (Pipeline Details) will have a deep look at each stage of the pipeline, explaining their working principle and documenting each key decision made during the development of the pipeline artifact.
- Section 11 (Evaluation) will evaluate the pipeline artifact through its runtime performance and the quality of its output.
- Section 12 (Concluding Remarks) will summarise the outcome of our research and provide guidelines for future research

Chapter 2

Background

This section will provide the essential concept of KGs and LLMs and define the specific subtype of KGs that we focus on in our study.

2.1 Various KGs

There are many variations of KGs (Hogan et al., 2020), as shown in Figure 2.1. For example, the KG that follows The Resource Description Framework (RDF), uses the (Subject, Predicate, Object) triples to represent a single piece of knowledge (Hogan et al., 2020). In contrast, the KG based on the Property Graph allows additional information stored in the predicate, making it beyond the traditional SPO triples (Hogan et al., 2020). The KG based on RDF* (also known as *RDF-star*), on the other hand, allows a triple itself to be embedded in another triple (Hogan et al., 2020). Consider the text "Amy was practising piano at 5:00 PM" as Figure 2.2 shows, with different KGs, the representation of the text can be also different. The KG based on Property

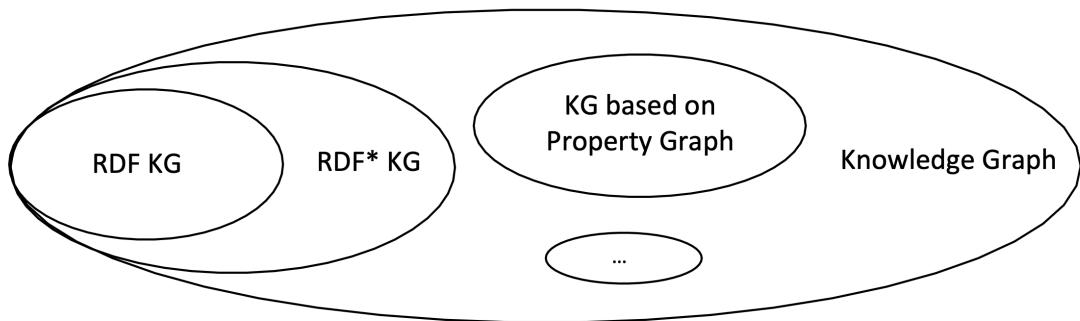


Figure 2.1: A Venn Diagram of Knowledge Graph Models

2 Background

Text: "Amy was practising piano at 5:00 PM."

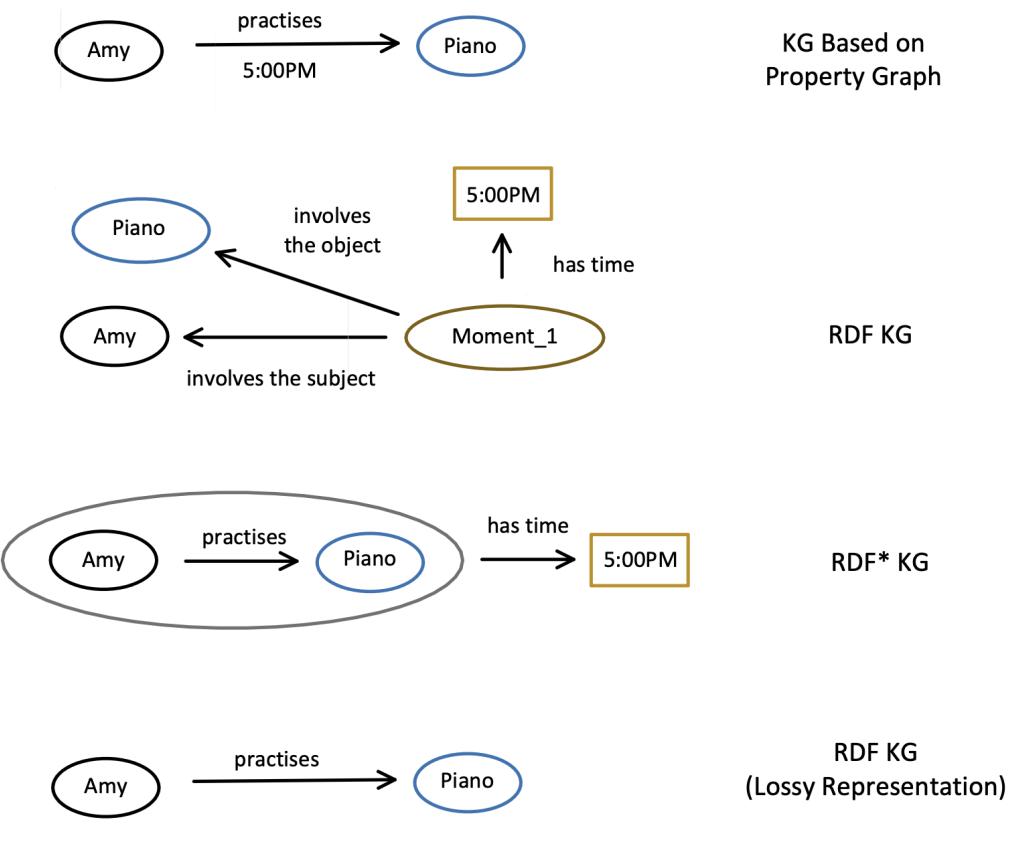


Figure 2.2: Different KG representations of a sentence

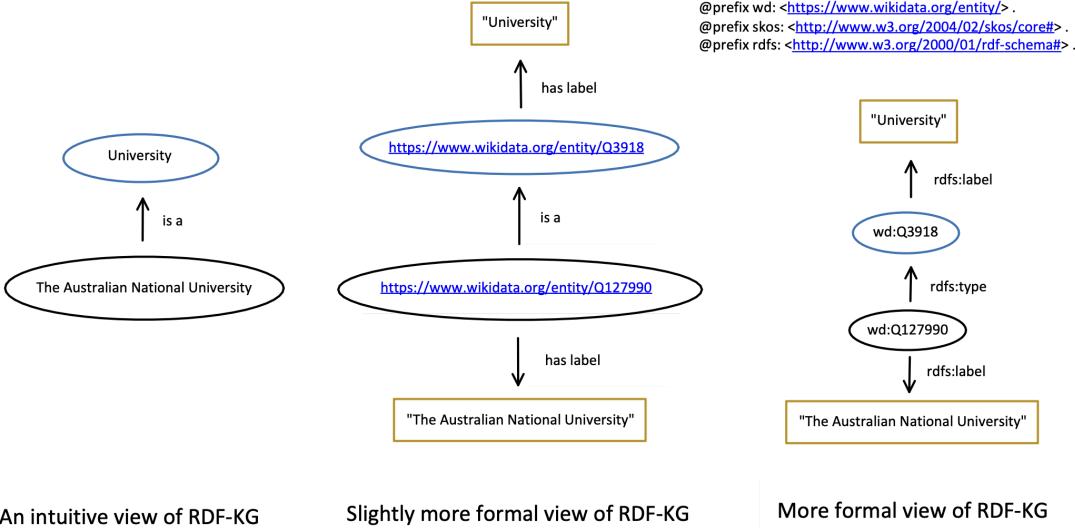


Figure 2.3: Different views of RDF-KGs

Graph can efficiently capture the knowledge using a single relation between two **Entity** nodes only, whereas the RDF-KG may have to introduce the extra **Event** node to store the temporal information (Hogan et al., 2020). Our study, however, focuses only on building RDF-KG. This is also the type of KG that *many recent studies* still focus on (Pan et al., 2023).

2.2 Some Properties of RDF-KG

2.2.1 Identity

One key property of RDF-KG that we have not mentioned in the introduction (Section 1.1) is the unique way of identifying each Entity node. Previously, we have put the name of an entity directly into its node (the oval) in our intuitive view of a KG, as Figure 2.3 (left) shows. However, what is actually stored inside an entity node and represents the entity itself is not a name but a global identifier called the Internationalized Resource Identifier (IRI) (Hogan et al., 2020; W3C, 2014), as Figure 2.3 (middle) or a local identifier (Hogan et al., 2020; W3C, 2014), whereas the name or alias of an Entity is usually stored outside an Entity node as a Literal node connected to the Entity. This is because the entity names, such as KG (Knowledge Graph) and KG (a unit), may not be unique. In fact, at the RDF graph level, there are only three types of nodes: the IRI node, the empty node (a node storing a local identifier), and the literal node (Hogan et al., 2020; W3C, 2014). At the RDF-KG level, an Entity is represented by either an IRI node or an empty node (Hogan et al., 2020). In our study, however, we only consider the case when Entities are represented by IRI. That is, we will assign each Entity that

2 Background

is extracted from a document by our pipeline an IRI instead of any local identifier as the representation of such Entity.

Note that the difference between IRI, e.g., <https://www.wikidata.org/entity/Q127990> and the Uniform Resource Locator (URL), e.g., <https://www.wikidata.org/wiki/Q127990> is that the URL usually represents the website describing an entity, whereas the IRI represents the entity itself (Hogan et al., 2020; W3C, 2014). Since Entities in a KG usually have the same IRI prefix, we can optionally pull out the prefix and give it a shortcut, e.g., wd. Then, only store the postfix of the IRI plus the shortcut, e.g., wd:Q127990, into the entity node (W3C, 2014) for space efficiency and clarity, as Figure 2.3 (right) shows. The IRI shown in Figure 2.3 is actually the IRI of Entities in the KG Wikidata (Wikidata, 2024). However, even if it is global, the IRI for the same entity in different KGs can be different. For example, the IRI for the Entity ANU in ASKG may start with <https://www.anu.edu.au/data/scholarly/> (Jia et al., 2024). In this case, Knowledge Graph Alignment is required to match different IRIs to the same Entity in the real world (Hogan et al., 2020).

Similarly, the predicates are also uniquely identified by IRIs (Hogan et al., 2020; W3C, 2014). For some predicates, such as `is a`, that are commonly used, there have been some schemas to define and formalise them (W3C, 2024; Hogan et al., 2020).

Throughout the thesis, we will continue using the *intuitive views* for conciseness, representing each entity by its label rather than showing the IRI.

2.2.2 Storage

RDF-KG is commonly stored using the Terse RDF Triple Language (TTL) as a `.ttl` file. In this format, only the edges, e.g., the triples, are explicitly stored (W3C, 2014, 2024; Hogan et al., 2020). The below shows a simple RDF-KG expressed in a TTL, containing the knowledge about (ANU, `is a`, university), (ANU, `is a`, public university), and (ANU, `located in`, Canberra). In addition to the feature of using prefixes as mentioned in 2.2.1, the triples with the same subject or the same subject and predicate in the TTL files may be grouped together for space efficiency.

```
@prefix wd: <https://www.wikidata.org/entity/> .  
@prefix wdp: <https://www.wikidata.org/wiki/Property:> .  
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .  
  
wd:Q127990 wdp:P276 wd:Q3114 ; # (ANU, located in, Canberra)  
      rdfs:type wd:Q3918, # (ANU, is a, university)  
      wd:Q875538 . # (ANU, is a, public university)
```

2.2.3 Flexibility

A KG, including RDF-KG, is highly *flexible* compared to other traditional data models, such as Relational Models (RM), that can also be used to represent knowledge (Hogan

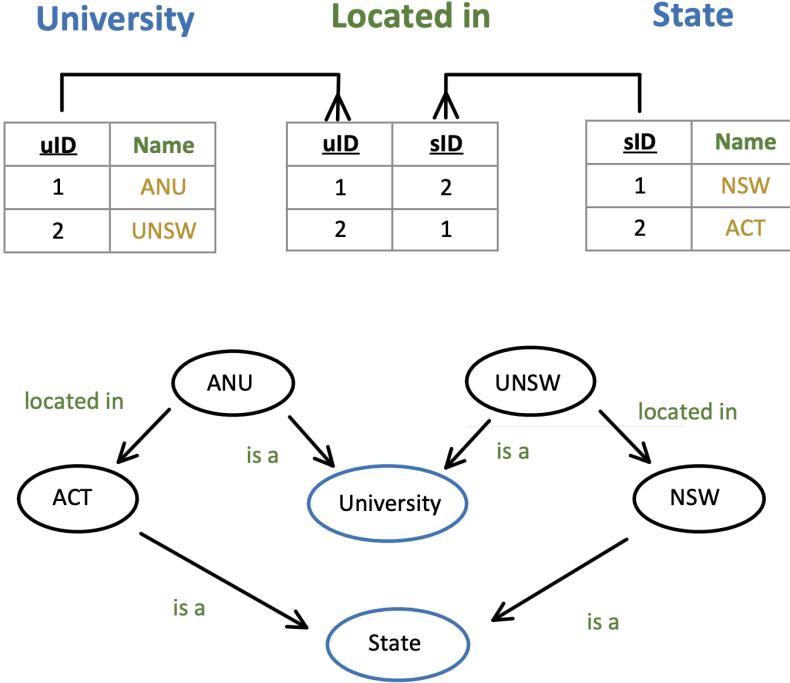


Figure 2.4: A KG and its equivalent relational model

et al., 2020). Figure 2.4 shows the comparison of using RDF-KG and Relational Model (RM) to represent the same knowledge about ANU, UNSW etc. In the RM, a Concept, such as `University`, also known as Entity Type, is represented as a table, whereas a Named Entity, such as `ANU`, also known as Entity Instance, is represented as a row or tuple along with its attributes (Elmasri and Navathe, 2015). The relations in RM may be represented as a column in the same table as the Named Entity, e.g. the `Name` column or a separate table. Note that, in the intuitive view of the KG (Figure 2.4), we have assumed that `ANU`, `UNSW`, etc., have been uniquely identified by some IRIs, not the local primary keys as in RM.

Compared to the relational model, RDF-KG is more capable of representing *incomplete* and *dynamic* knowledge.

For example, a named entity is allowed to have its type missing in RDF-KG (Hogan et al., 2020). In this case, there will not be the predicate, `is a`, between `ANU` and `University`. However, this does not necessarily mean that `ANU` belongs to Nothing. It just means we do not know the exact types of `ANU`. Therefore, RDF-KG is typically under the Open World Assumption (OWA), where the missing of a statement like (`ANU`, `is a`, `University`) does not mean that the statement is False (Hogan et al., 2020), which is suitable for handling *incomplete* knowledge. In the relational model, however, we must explicitly assign `ANU` a type, whether a narrow type like `University`

2 Background

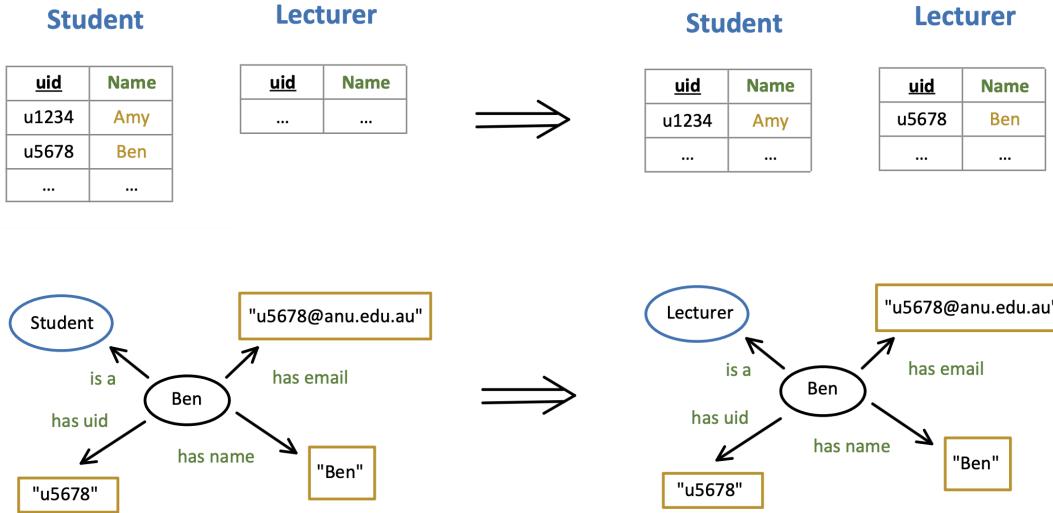


Figure 2.5: Dynamic of data

or a broad type like `Organisation`. This is understandable because, in a general sense, a single row (without a column header) cannot exist independently and meaningfully unless being part of a table. Therefore, in RM, a Concept must exist before its Named Entities exist. There is a clear distinction between Named Entities and Concepts, whereas, in RDF-KG, everything is just an independent node or edge.

KG is also good at handling dynamic data. As Figure 2.5 shows, a `Student` called Ben in a university may become a `Lecturer` many years later. In KGs, this can be as simple as redirecting the edge, `is a`, from `Student` to `Lecturer`. Whereas in the relational model, we may have to move all Ben's information, including the `Name` from the `Student` table to the `Lecturer` table if there is not predefined supertype table like `Faculty` to store the common attributes, like `Name` between `Student` and `Lecturer`. Even though in such a case, we still need to at least move Ben's primary key.

These two properties will be mentioned again in Literature Review (Section 3.3) when discussing potential reasons for the limitations of traditional KGC. A small hint is that traditional KGC tends to be constructed following steps similar to those used in traditional data models, like the relational model. However, due to its flexibility, the construction of a KG can also be more adaptable, leading to a more holistic KGC.

2.3 Definition of Entity, RDF-Graph, RDF-KG

While the previous sections aim to give an intuitive introduction to KG and RDF-KG. This section aims to give a more formal definition of RDF Graph and RDF-KG.

2.3.1 Entity and Literal

Although [Hogan et al. \(2020\)](#) in their book Knowledge Graph...

The term Entity can be vague and defined differently across the literature. Therefore, we will follow the definition in the Oxford Dictionary ([Oxford University Press, 2024](#)).

Definition 1 (Entity). An Entity is something that exists separately from other things and has its own identity. ■

Therefore, an entity can technically be anything as long as it has an identity. Hence, in this study,

- we would consider *anything* that is identified and represented by an IRI node or an empty node in an RDF-KG as an Entity. An Entity may include Named Entity (a specific thing, e.g., Amy), Concept (an abstract or conceptual thing, e.g. Piano), Event (e.g., Amy's Piano Practising Session), etc. They have been and will be shown as oval in our graphs. However, in this study, we will focus on *Named Entity* and *Concept*.
- We would consider *anything* that is represented by a Literal node as Literal rather than an Entity because it does not have its own identity. It cannot exist independently and meaningfully without being an attribute of some entities. They have been and will be shown as *rectangles* in our graphs.

2.3.2 RDF-Graph

An RDF graph is a graph-based data model to represent structured data as a set of triples. It can be modelled as a directed labelled graph in mathematics ([W3C, 2024](#)). Each triple represents a directed relation between a subject and an object with a predicate ([W3C, 2024](#)), in formal,

Definition 2 (RDF Graph). An RDF graph G is a set of triples, where each triple is represented as (s, p, o) , where:

$$G = \{(s_i, p_i, o_i) \mid s_i \in U \cup B, p_i \in U, o_i \in U \cup B \cup L\}$$

with:

- U : the set of all Internationalized Resource Identifiers (IRIs),
- B : the set of all blank nodes,
- L : the set of all literals.

with:

- IRI: a globally unique identifier for some resource, such as an Entity.
- Blank Node: a non-global identifier for some resource, such as an Entity.
- Literal: a literal value, such as a string or number.

■

Therefore, in RDF-KG, the one in the subject position in a relation must have an identity, either an IRI or a local identifier. During our KGC, we will also treat attributes as an entity, i.e., storing them using an IRI node instead of literal nodes, as long as they appear in the original text. Whereas the literal nodes is mostly used for storing the aliases and description of an entity only. This is because the distinction between entity and attributes can also be vague. Consider the following sentence:

"ANU was founded in 1946. 1946 is a special year."

Although 1946 can be considered an attribute of ANU, if we store it using a literal node, we will no longer be able to model the next triple, i.e., (1946, is a, special year) unless creating another 1946 using an IRI node. This is because a literal node cannot be a subject. Therefore, for simplicity, we will use the IRI node to represent everything that syntactically appears in the document and is extracted by us as an entity.

2.3.3 RDF-KG

Clearly, not every RDF Graph should be considered RDF-KG in a general sense because an RDF Graph only defines a way for storing data but not semantics. However, the boundary between whether an RDF Graph can be considered a KG is unclear and potentially context-dependent, and even the term "knowledge" can be defined in many ways.

For example, an RDF Graph with nodes representing some binary data may more or less convey the "knowledge" about the relationship between these binary data. However, it may not convey any deep semantics directly connecting words and phrases. Similarly, our inputs, the semi-structured academic papers represented by RDF Graphs, indeed convey the semantics or "knowledge" about the structures and metadata of papers, and we may refer to them as RDF-KGs of the metadata of the papers.

Therefore, we will not provide a definition for an RDF-KG. However, we would define *RDF-KG of text documents* as the following:

Definition 3 (RDF-KG of text documents). An RDF-KG of text documents is an RDF-based KG representing the semantics of the content of one or more documents of any length. ■

From now on, we would

- Only refer to graphs that represent the content of text documents as **Knowledge Graphs**.
- And refer to our inputs, the semi-structured papers in RDF-Graph form, as **Metadata Graphs**.

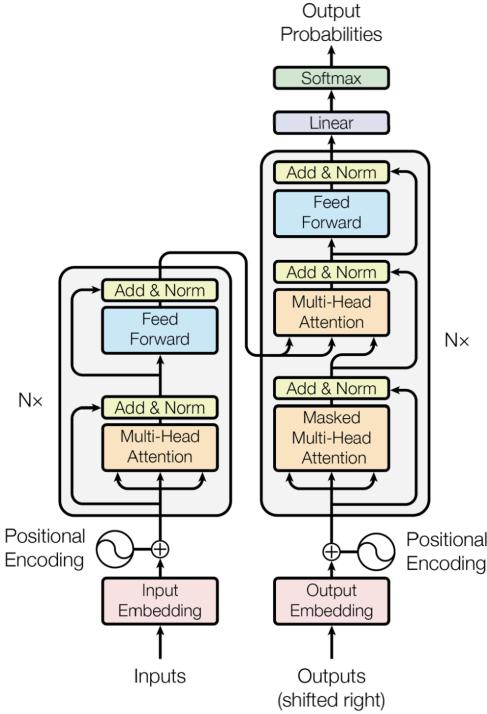


Figure 2.6: The transformer model architecture ([Vaswani et al., 2017](#))

2.4 LLMs

In general, there are three types of LLMs, the LLMs based on the *encoder*, *decoder*, and *encoder-decoder* architecture of the transformer model ([Pan et al., 2023](#)). Figure 2.6 shows the basic structured or transformer model. The left is the encoder, and the right is the decoder, whereas the combination of the left and the right models is the encoder-decoder.

However, in this thesis, we focus only on LLM encoders and LLM decoders, as they are vastly applied in KGC and have been integrated into our final KGC pipeline artifact. For simplicity, we will treat LLMs as a *black box* in this study without delving into their detailed internal architectures.

2.4.1 Encoders: Training, Inference, and Complexity

In general, an LLM encoder takes a piece of text in natural language as input (in the form of a list of tokens) ([Pan et al., 2023; Vaswani et al., 2017](#)). It generates a compact vector representation, called embedding, for each input token in the original text ([Vaswani et al., 2017](#)). These embeddings are not only generated based on the surrounding context of each token but also the model's learned understand-

2 Background

ing of natural language (Vaswani et al., 2017). For example, consider the sentence, "Ben said he will finish the thesis today.". The embedding of "he" may not only tell that this is a pronoun based on the model's general language understanding but also tell that it refers to Ben. However, such a telling is encoded in vector form, and the training of a classifier is usually required to decode the meaning of these embeddings to perform downstream tasks (Pan et al., 2023). For example, in Named Entity Recognition using encoders, the embedding for each token will then be converted into some human-readable labels, such as Person, Location and 0, to tell whether a token is a Named Entity or not and, if so, what is its type (Li et al., 2021).

LLM encoders are typically trained using a process called *masked language modelling* (Vaswani et al., 2017; Pan et al., 2023). During training, the encoder receives a large corpus of text, with some tokens masked out in each input sequence. The encoder then attempts to generate the embedding for each input token in the original text, including those that are masked out. For those masked tokens, the encoder has to understand the semantics of the original text so that it can generate a suitable embedding that represents the potential original token based on other tokens that are not masked. The final classification layer then converts these embeddings for the masked token into the predicated tokens. By comparing the predicted tokens with the actual masked tokens, the model can know if its generated embedding is good enough so that it can adjust its parameters. As training progresses, the encoder gradually improves its ability to understand and represent complex language patterns and nuances using its embeddings. (Vaswani et al., 2017; Pan et al., 2023)

During inference, the complexity of an encoder is largely dominated by its Multi-Head Attention as shown in Figure 2.6. In this stage, every input token needs to be compared with others, leading to a rough complexity of $O(n^2)$, where n is the number of tokens in the original sentence (You et al., 2024; Zhou et al., 2024).

2.4.2 Decoders: Training, Inference, and Complexity

An LLM decoder takes a piece of text as input and predicts the next token that likely follows the input sequence (Pan et al., 2023; Roberts, 2024). When used in an autoregressive manner—where the generated token is appended to the original input, and the model is called iteratively to predict successive tokens—the decoder can generate entire sentences or paragraphs that logically follow the input (Pan et al., 2023; Roberts, 2024). This autoregressive generation enables decoders to perform many generation tasks, such as answering questions or generating chatbot responses (Pan et al., 2023).

Decoders are typically trained using *causal language modelling* (Roberts, 2024). During training, the decoder is fed partial sequences from a large text corpus and is trained to predict what comes next based on the given context. The full-text sequence is available as a reference after its prediction, enabling the model to adjust its parameters to improve future predictions. This process inherently teaches the decoder the underlying semantics of the text, as understanding context is necessary for accurate next-token prediction.

After training, the decoder can then generate meaningful continuations of input text. (Roberts, 2024)

During inference, when a decoder is used to predict only one token after the original text, the complexity is similar to decoders, which is also roughly $O(n^2)$ as decoder also involves the Multi-Head Attention mechanism as shown in Figure 2.6 (You et al., 2024). However, since the encoder is often used autoregressively to generate a list of tokens, the complexity can become cubic.

To generate a text consisting of k tokens given an input text, the model has to be called k times where every time the model is called, the length of the input text is increment by 1. Therefore, assume that the length of the initial text (also known as prompt) when the model is first called, is m , and k is the number of new tokens generated, i.e., the number of tokens in its response. Then $m + k = n$ is the number of tokens of the initial prompt plus its final response.

Then, the cumulative complexity for generating k tokens with input length m is:

$$\sum_{i=m}^{m+k} O(i^2) = O\left(\sum_{i=m}^{m+k} i^2\right)$$

Given the sum of the squared formula

$$\sum_{i=1}^n i^2 = \frac{n(n+1)(2n+1)}{6}$$

and

$$\sum_{i=m}^n i^2 = \sum_{i=1}^n i^2 - \sum_{i=1}^{m-1} i^2$$

then

$$\sum_{i=m}^n i^2 = \frac{n(n+1)(2n+1)}{6} - \frac{(m-1)m(2m-1)}{6}$$

Assume that the *response is about the same length as the prompt*, which is plausible when people ask an LLM decoder, like ChatGPT, for solving questions, i.e., $m = k$ and thus $n = 2m$, then

$$\begin{aligned} \sum_{i=m}^{2m} i^2 &= \frac{(2m)(2m+1)(4m+1)}{6} - \frac{(m-1)m(2m-1)}{6} \\ &= \left(\frac{4m^3}{3} + m^2 + \frac{m}{3}\right) - \left(\frac{m^3}{3} - \frac{m^2}{2} + \frac{m}{6}\right) \end{aligned}$$

2 Background

$$= m^3 + \frac{3m^2}{2} + \frac{m}{6} = O(m^3)$$

Since, $n \approx 2m$, the complexity dependent on the length of the prompt plus response become $O(n^3)$

Therefore, it can be found that LLM decoders are generally less efficient than LLM encoders.

2.4.3 Encoders and Fine-Tuning

Using an encoder often requires fine-tuning to adapt it for specific downstream tasks unless a simple cosine similarity comparison of raw embeddings is sufficient (Liu et al., 2024). This is because the embedding for the input text is generally human-unreadable, and we may need a task-specific layer, such as a classifier, to translate it into task-specific outputs, such as labels (Liu et al., 2024). Fine-tuning generally requires datasets, but the use of datasets may limit the LLM’s ability to only generate labels that fall within a predefined set, which reduces its ability to be adapted to new domains or handle unseen inputs and edge cases (Liu et al., 2024). For example, a Named Entity Recognition system based on an LLM encoder fine-tuned on some academic datasets may only recognise the academic term in a text as named entity mentions. This limitation will be mentioned again in Section 3 when discussing the potential reason for the limitation of traditional KGC.

2.4.4 Decoders and In-Context Learning

On the other hand, Using an LLM decoder can often be more straightforward than using an encoder, as it typically does not require additional layers, such as classifiers, to make its output interpretable (Liu et al., 2024).

However, since there is no fine-tuning involved, we then need to put our instructions, along with the input data that we want to process, such as a text from which we want to extract entities, into the prompt in order for the model to learn what it is supposed to do to finish a task. Such a way of using LLMs is also called In-Context learning (Sahoo et al., 2024). We make the model “learn” what it should do using the input prompt instead of tuning its internal parameter of the model, unlike what is usually done for LLM encoders (Sahoo et al., 2024).

Since the model functionality and performance will largely depend on the prompt, writing a good prompt becomes essential. Therefore, this also led to a study of how to write prompts to make LLM decoders perform effectively, known as *prompt engineering* (Sahoo et al., 2024).

Common techniques in prompt engineering include:

- **Zero-shot prompts:** Asking the model a question or giving it a task directly without prior examples (e.g., “Extract all Named Entities in the sentence {sentence}”)

(Radford et al., 2019).

- **Few-shot prompts:** Provide examples within the prompt to help the model generate responses that follow a desired format (e.g., “Extract all Named Entities in the sentence {sentence}. Here are the example inputs and outputs: {examples}”) (Brown et al., 2020). Usually, these examples are selected from the training dataset by matching the actual data that we want the LLM to process with the relevant input-output examples in the dataset (Brown et al., 2020). However, given such a dataset may not be always available, we can also provide some fixed hand-made examples along with our prompt. Even with a few examples, the LLM decoder should be able to learn some patterns from them (Roberts, 2024).

A side note is that the use of the LLM decoder often involves the question-answer pattern. That is, we want the model to answer our questions or perform tasks based on our instructions in the prompt instead of generating anything that naturally continues our original prompt. For example, given the input "What is the capital city of Australia?", a raw decoder might produce a response like "This is a good question." rather than a direct answer like "Canberra". Therefore, what is commonly used in practice is the Chat version of the original decoder that specialises in answering user questions or completing users' instructions. (Liu et al., 2024)

2.4.5 Complexity Issue in Decoder

Although the LLM decoder can be considered more flexible, it does not require training or fine-tuning of datasets. It also has some critical issues.

As discussed before, the complexity of LLM decoders is internally higher than LLM encoders, $O(n^3) > O(n^2)$. Not only that, we have to put both the data to be processed and the prompt as our input, like when using encoders which often take the data only. Therefore, the term n is typically larger than the n in encoders for achieving a similar task, which can make the runtime even worse. In fact, during the implementation of our pipeline, we have seen that the pipeline stage that Later on, we will see that the pipeline stage that involves the encoder can only be done within a second, even when processing a batch of text., which will be discussed in detail in Section 11.

2.4.6 Hallucination Issues in Decoder

LLM decoders also suffer from an issue called hallucination. It tends to always generate something even if there are no answers to the prompt. For example, if the user asks, “When did Newton invent the light bulb?”, the LLM may still answer, “Newton invented the light bulb in 1879” (Pan et al., 2023). The hallucination can be due to reasons. One of the key reasons is that the LLM decoders do not inherently know and are able to validate facts (Pan et al., 2023). It just generates anything that possibly continues the original text, but there is no guarantee that the text generated is corrected (Pan et al., 2023).

Chapter 3

Literature Review

This section will cover the traditional approach to KGC. Then, it will show how the traditional KGC approach can be implemented by LLMs. Following that is a summative analysis of the limitations of traditional KGC. The section will end with providing more recent work that tackles one or more limitations of traditional KGC.

3.1 Traditional KGC Approach

[Pan et al. \(2023\)](#); [Zhong et al. \(2023\)](#); [Hogan et al. \(2020\)](#); [Ye et al. \(2022\)](#)'s studies have collectively provided a general approach for constructing KGs from text documents. Many studies we found tend to follow a similar pattern, although the detailed implementation can be different ([Pan et al., 2023](#); [Xie et al., 2023](#); [Ashok and Lipton, 2023](#); [Wang et al., 2023](#)).

The general KGC approach from documents can be divided into several stages, as shown below.

1. Schema Definition.
2. Named Entity Recognition.
3. Entity Typing.
4. Entity Disambiguation.
5. Coreference Resolution.
6. Relation Extraction.

Note it is possible to have more stages other than the ones listed, such as Event Extraction ([Ye et al., 2020](#)). However, since we mainly focus on constructing KG representing

3 Literature Review

the relationship between Named Entities and Concepts, as mentioned in Section 2.3.1, we will only consider the six as shown.

3.1.1 Schema Definition

The first step is to a schema for the KG. This involves determining the types of named entities that are going to be extracted and the predicates (Hogan et al., 2020). This sometimes requires human intervention to examine the potential types of entities within the set of documents that are going to be converted into KG (Hogan et al., 2020). However, in practice, the schema is usually set by the dataset that is used to train a KGC model, such as a LLM. Such datasets, e.g., SciERC (Luan et al., 2018) and DocRDE (Yao et al., 2019), have provided the mappings from text documents to their corresponding KG, and these datasets typically have their predefined schema about the types and relations that can exist in the final KG (Luan et al., 2018; Yao et al., 2019). Therefore, this step is more about choosing an appropriate dataset for training an LLM KGC model. For example, to construct KG from academic papers, we may choose SciERC, which contains the academic types, such as `Method` and `Task` and predicates, like `USED-FOR` (Luan et al., 2018), so that the constructed KG can focus on capturing academic knowledge. For building KG from general articles, we may choose DocRDE, which contains general types like `PERSON` and `LOCATION` (Yao et al., 2019). Figure 3.1 shows a simple example of the Schema Definition step. In this example, we focus on capturing the entity type `Organisation`, `Person`, and `Location`, and the predicates `located in`, `head of`, and `is a`. This step can be an analogy to creating tables in traditional relational models.

3.1.2 Named Entity Recognition and Entity Typing

The next step, Named Entity Recognition, is to identify and extract all the Mentions of Named Entities from a document (Pan et al., 2023; Zhong et al., 2023; Hogan et al., 2020). Mention in KGC refers to a phrase in a text that represents some entity (Hogan et al., 2020). It can be explicit, like a proper noun ANU or implicit, like a pronoun It (Hogan et al., 2020).

Since this step is about extracting mentions and not the entities themselves. To be clearer, we would refer to it as Named Entity Mention Extraction since some entities can only semantically appear in a document but not syntactically appear as a mention, which will be discussed in Section 4.4.2. Mention should be considered a linguistic concept, while Entity is a real-world concept. They are at different layers.

Traditional NER models tend to only extract explicit mentions of Named Entities (Xie et al., 2023; Ashok and Lipton, 2023; Wang et al., 2023) only. A simple Entity Typing, usually along with Named Entity Recognition, assigns the Named Entity to extract one of the predefined types (Xie et al., 2023; Ashok and Lipton, 2023; Wang et al., 2023). This is because for a Named Entity to be extracted, it must fall within one of the predefined types. However, more advanced **Entity Typing** can involve assigning

3.1 Traditional KGC Approach

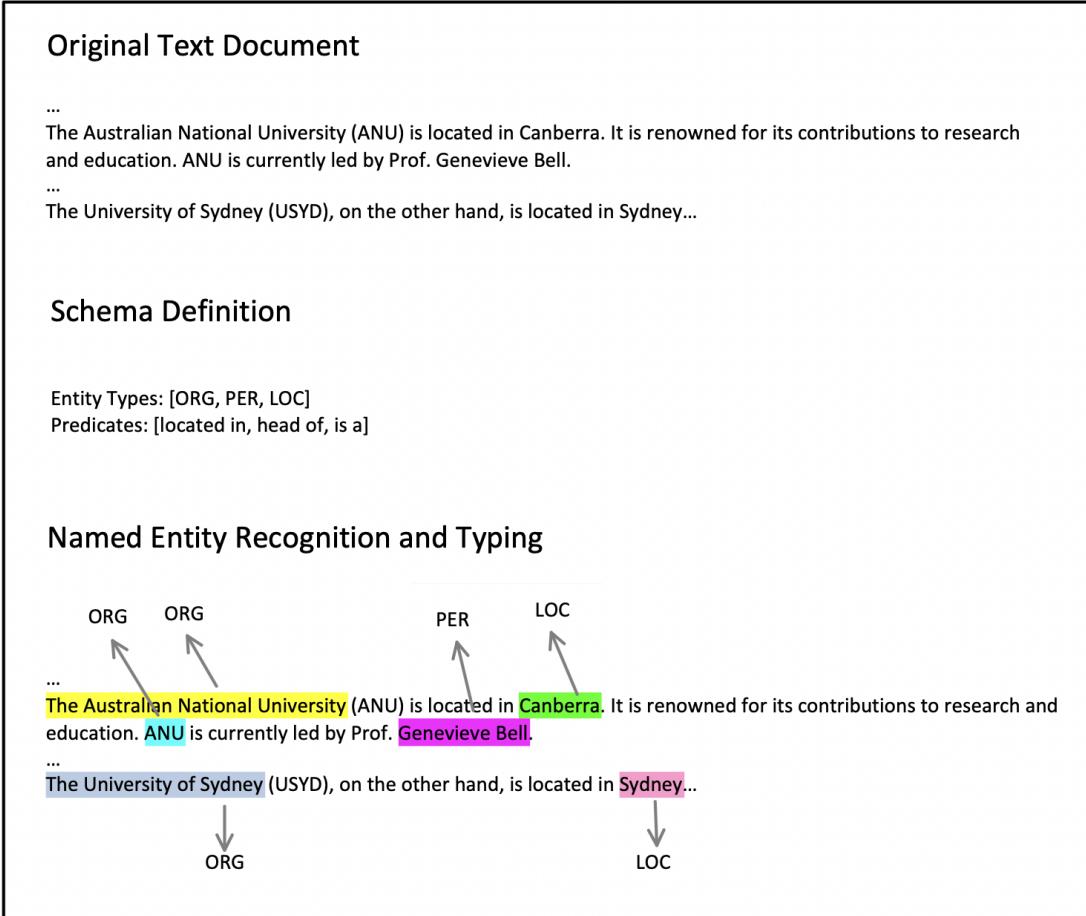


Figure 3.1: Traditional KGC: Process 1-3

fine-grained entity type to a named entity (Hogan et al., 2020). For example, the named entity ANU may be classified as a **Public University** instead of just an **Organisation**. Figure 3.1 shows a simple example of how traditional NER can be performed.

3.1.3 Coreference Resolution and Entity Disambiguation

In a document, entity mentions with different names can refer to the same named entity. For example, the mentions **The Australian National University** and **ANU**, as shown in Figure 3.2. Meanwhile, there can be multiple persons in the world who have the same name, e.g., Jack. Therefore, **Coreference Resolution** is required to group different mentions of the same named entity into the same named entity node in a KG, whereas **Entity Disambiguation** involves distinguishing two identical mentions referring to different things to avoid grouping them together (Hogan et al., 2020; Pan et al., 2023). The two processes may be done together or separately (Pan et al., 2023) since both may

3 Literature Review

involve creating a context-sensitive understanding for each entity, i.e., an embedding, and with these, it can be easy to know whether the two mentions refer to the same thing or no matter if their actual names are different or not. Figure 3.1 shows how Coreference Resolution and Entity Disambiguation are performed in a simple manner.

3.1.4 Relation Extraction

Relation Extraction involves extracting the relations between entities by connecting them with some predicates (Hogan et al., 2020). Traditionally, these predicates are usually predefined in the very first step (Pan et al., 2023). Relations may be extracted from a sentence only or from the whole document (Hogan et al., 2020; Pan et al., 2023), although document-level Relation Extraction can sometimes be challenging. Figure 3.1 a simple example of Relation Extraction. After this process, the task of KGC from documents can be considered done, although there is a process called Knowledge Graph Completion, which aims to further improve the quality of a constructed KG, e.g., finding missing relations through techniques based on the constructed KG only, instead of the look back to the original documents Pan et al. (2023); Hogan et al. (2020). However, this is usually considered another field of study (Pan et al., 2023).

3.2 LLM-Based Implementation of Traditional KGC

Since the introduction of the KGC approach, many studies have followed this traditional KGC approach using LLM (Pan et al., 2023). In this study, we will first show how the stages in traditional KGC can be implemented using both LLM encoders and decoders in an intuitive and simplified way. Then, in the last part of this section 3.2.3, we will show some more advanced LLM-based traditional KGC implemented developed from the baseline implementation with higher performance, for example. However, they still follow a similar pattern as traditional KGC, such as using predefined types and relations. Although the actual implementation can vary across different literature, the following aims to show the baseline approach to such implementation.

3.2.1 Named Entity Recognition and Entity Typing

Figures 3.3 and 3.4 shows how the **Named Entity Recognition and Typing** in the traditional KGC process are performed using LLM encoders and decoders at a high level. On the encoder side, the LLM encoder first reads and understands the given text and then, for each token, provides a contextualised embedding (Devlin et al., 2019; Li et al., 2021). This embedding will implicitly tell whether or not this token is part of a named entity and the type of such a named entity (Devlin et al., 2019). Afterwards, a classifier will turn the embedding into human-readable types, and hence, the results are obtained (Li et al., 2021). Training a classification head is required when using LLM for NER because the LLM encoder does not inherently know how to provide the type labels. The job of the classifier is to within the relevant semantic information from the embeddings to determine whether the token is a named entity and, if so, what its type, Li-2021.

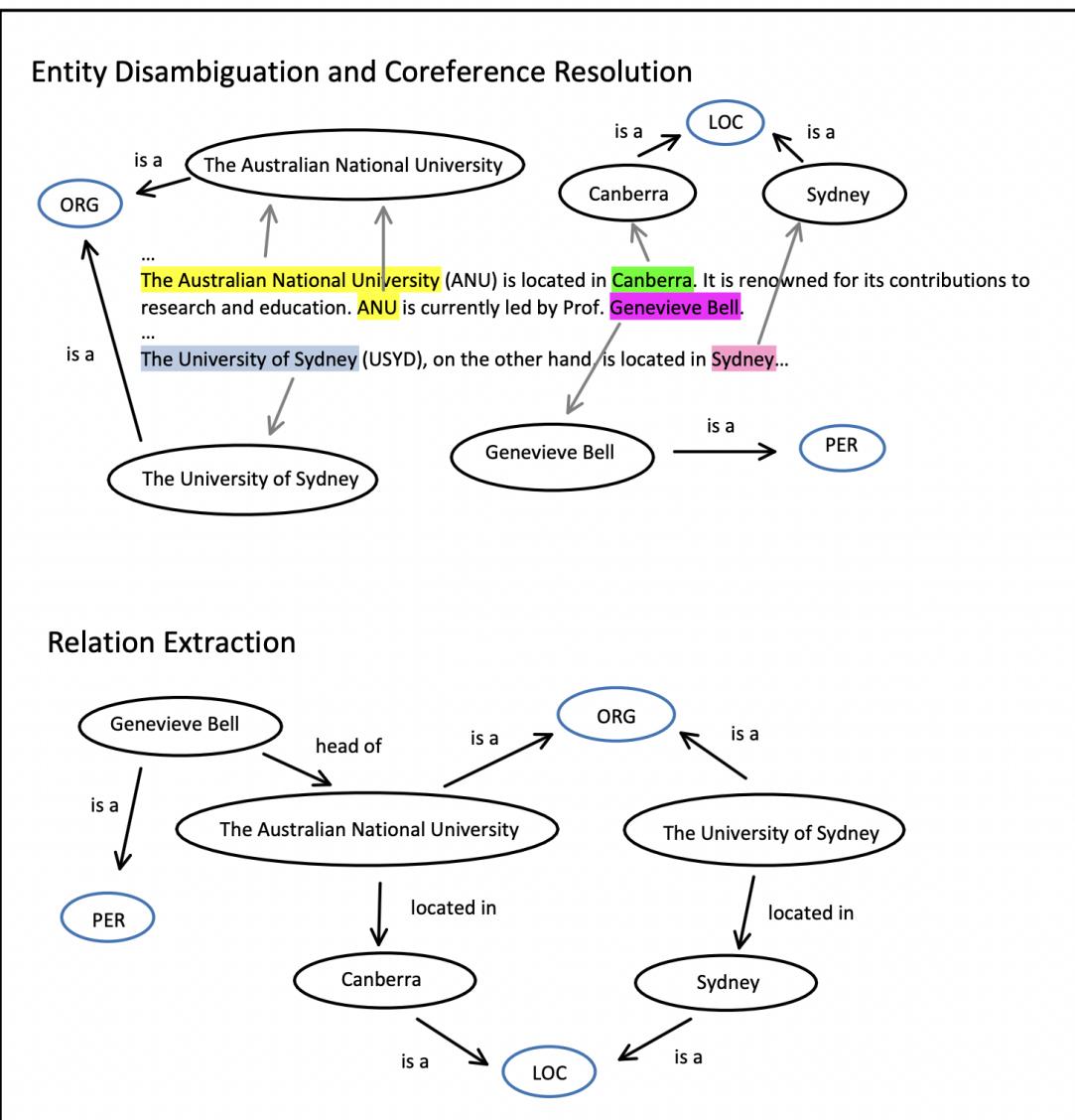


Figure 3.2: Traditional KGC: Process 4-6

3 Literature Review

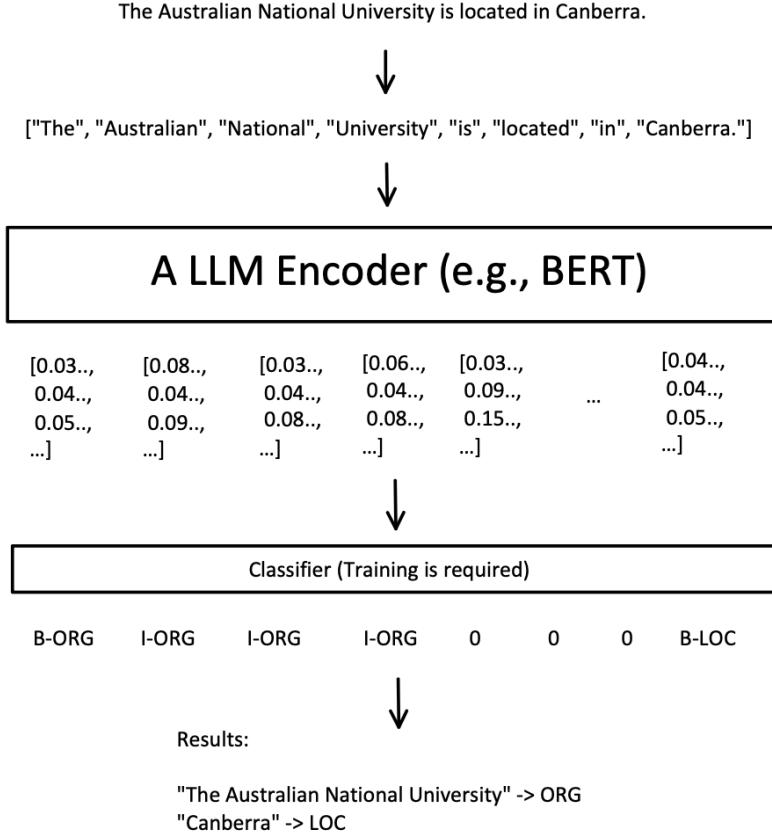


Figure 3.3: Named entity recognition and typing using an encoder

Since the use of an LLM decoder does not rely on examples and training, which can limit the LLM to only being able to retrieve named entities belonging to the pre-defined types, we can change the phrase “Extract all named entities belonging to the LOC and ORG types” into “Extract all named entities and assign them some types”. In this way, the LLM decoder can then extract anything it thinks is a named entity based on its language understanding and the provided context. This is exactly what we want to achieve in our holistic KGC.

The LLM decoder methods, on the other hand, do not need the training process (Xie et al., 2023; Wang et al., 2023); in general, explicitly asking the LLM decoder to extract named entities that fall into the predefined types can obtain some results from the LLM (Xie et al., 2023; Wang et al., 2023). Optionally, we may provide some examples of the mappings between a sentence and a list of named entities in the sentence during the inference stage, instead of the training stage like encoders, to improve its output quality (Xie et al., 2023; Wang et al., 2023). Since the use of LLM decoders does not need to train a classifier like in the LLM encoder, the use of an LLM decoder in NER

3.2 LLM-Based Implementation of Traditional KGC

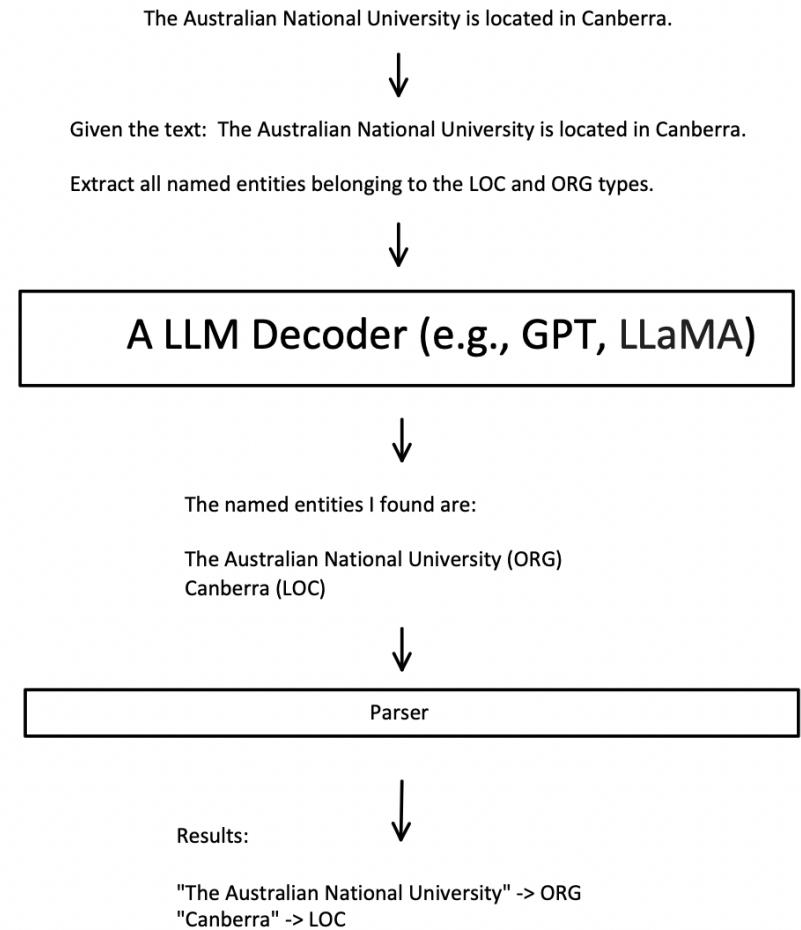


Figure 3.4: Named entity recognition and typing using a decoder

3 Literature Review

can be more flexible and is not confined to the predefined types listed in the example dataset(Wei et al., 2024).

3.2.2 Coreference Resolution, Entity Disambiguation, and Relation Extraction

For the differences between using LLM encoders and decoders to perform Coreference Resolution and Entity Disambiguation and Relation Extraction. We have provided the diagram in Figures 3.5 and 3.6 (Joshi et al., 2019; Zhang and Soh, 2024; Shi and Lin, 2019; Wei et al., 2024). A similar pattern is followed compared to the Named Entity Recognition and Typing task. However, there are some key points to be noticed.

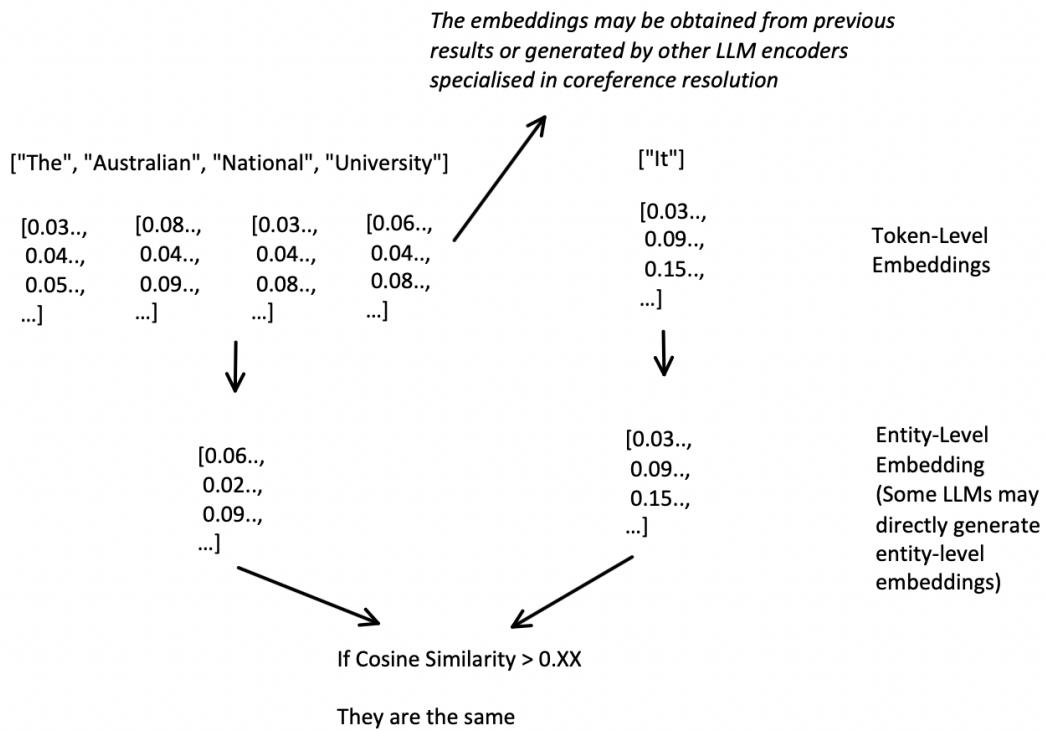
- The conference is done by comparing entity-level embedding. There are many ways to get entity-level embedding from token-level embedding, such as pooling (Joshi et al., 2019).
- The conference resolution part for encoders may not require training a classifier, as directly computing the dot product between two embeddings is enough to know if they are similar or not. However, fine-tuning the LLM can still help the LLM to specialise in Coreference Resolution. Adding a classification head also helps extract key information from embeddings so that the determination of whether two entities are the same can be more accurate (Joshi et al., 2020).
- As Relation Extraction using encoders requires training using examples, and examples typically already have a limited set of predicates predefined, it can limit the Relation Extraction process to only able to extract relations that fall into the predefined predicate list (Shi and Lin, 2019).

3.2.3 Literature Based on Traditional KGC

There are many studies that fall into the category of traditional KGC but have developed a model that is more advanced than the baseline, as shown above, to improve the overall performance.

Named Entity Recognition and Typing: Xie et al. (2023) have developed a Named Entity Recognition model based on the LLM decoder ChatGPT with the additional feature of Syntactic Prompting and Tool Augmentation. Instead of only putting the instruction as the original sentence as inputs, Xie et al. (2023) also provides the Part-of-Speed tagging and Chain-of-Thought instruction to let the model extract entity not only based on the semantics of the sentence but also the static Part-of-Speed taking. Wang et al. (2023) has developed an NER model that combines an LLM encoder and an LLM decoder with few-shot learning. In Wang et al. (2023)'s model, the Named Entity was first extracted first by an encoder. Then, these extracted named entities are matched with a dataset providing sentence-entities mapping examples. The retrieved relevant examples are then used as input along with the original sentence to allow to the decoder model to perform better in NER.

Coreference Resolution and Entity Disambiguation (by Encoders)



Coreference Resolution and Entity Disambiguation (by Decoders)

Given "The Australian National University" in Sentence A, and "It" in Sentence B, determine if they refer to the same entity.

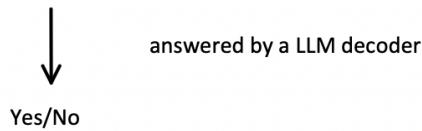
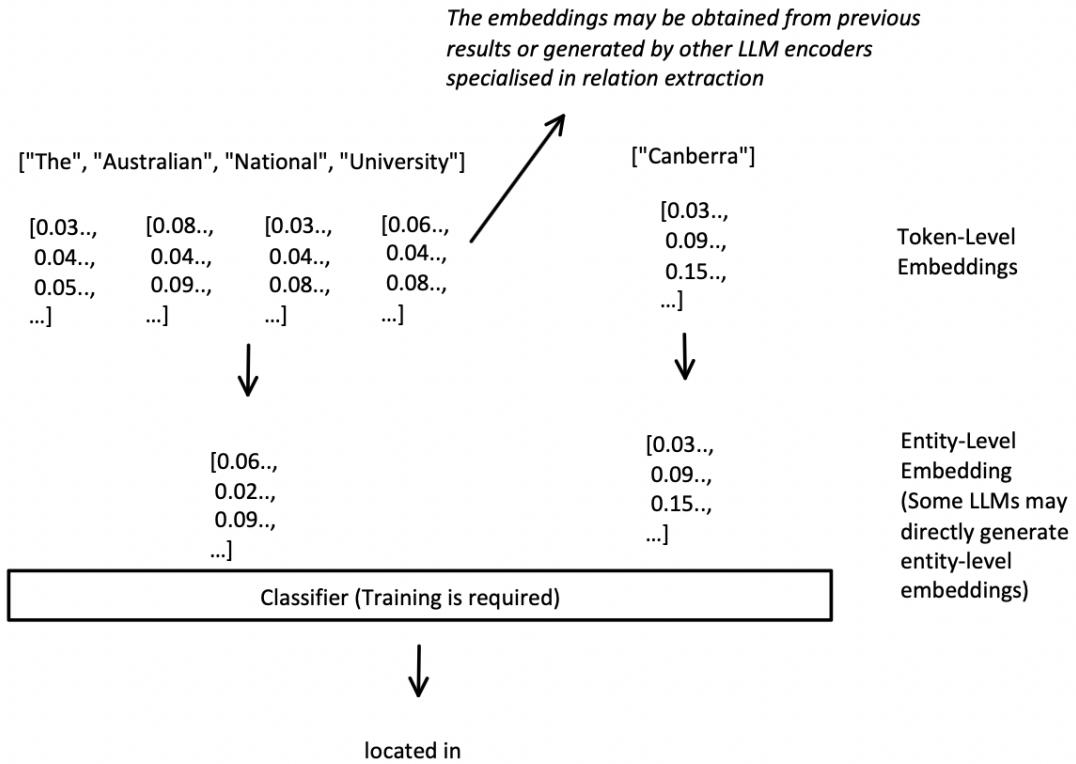


Figure 3.5: Coreference Resolution and Entity Disambiguation using encoders and decoders

Relation Extraction (by Encoders)



Relation Extraction (by Decoders)

Given the sentence "The Australian National University is located in Canberra." and the entities "The Australian National University" and "Canberra", what are the relationship can be found among these entities.

("The Australian National University", "located in", "Canberra")

Figure 3.6: Relation Extraction using encoders and decoders

Coresference Resolution and Entity Disambiguation: Lee et al. (2018) has outlined a general technique in Coresference Resolution, called coarse-to-fine, where rough embedding and scoring are first assigned to each mention. For those who are potentially close, more complex embedding generation techniques are used to generate fine-grained embedding to determine if two mentions refer to the same thing. In addition to using the mention itself and context for generating the embedding for coresference resolution and entity disambiguation, Li et al. (2020), Ayoola et al. (2022), and Salman et al. (2024) also consider the description and types of each entity mention, leading to higher performance.

Relation Extraction: instead of directly determining if two entities are related, Lyu and Chen (2021) developed a Relation Extraction model that applies type restriction. This method can improve overall precision by filtering out the predicates that are not type-compatible for entities. For example, ANU may not be the subject in the relation `graduate from` as the subject should be a person instead of a university. Zheng and Chen (2023) extend Lyu and Chen (2021)'s method by also using the description of entities during Relation Extraction. Tang et al. (2020) introduced multi-level Relation Extraction where both entity-level, sentence-level, and document-level information are used to generate embedding for entities when using encoders for Relation Extraction. In this way, the relations extracted can involve both sentence-level and document-level relations.

3.3 Reflection on Traditional KGC

There may already be some hints about the limitations of traditional KGC. In this section, we will highlight the limitations of traditional KGC and analyse the potential reason for the limitations.

3.3.1 Training Involved

Many studies for KGC focus on using encoder models. However, using an encoder typically requires additional training or a fine-tuning process. Fine-tuning can still consume a significant amount of computational resources, especially when the parameter number of LLM grows larger and larger these days. For example, from 334M's BERT-Large (Devlin et al., 2019) to 70B's LLaMA 3 (Meta, 2024). Also, if a KGC pipeline is built upon a fine-tuned LLM instead of the original LLM, the pipeline can become more dependent on a specific LLM. If more advanced LLMs are available, training or fine-tuning may need to be performed again to upgrade the pipeline. Pipelines built upon LLM decoders, on the other hand, are less dependent on the original LLM since the way of using LLM decoders, i.e., in-context learning, does not change its internal parameter (Sahoo et al., 2024) but with prompts. In this case, the pipeline may quickly adapt to new LLMs.

The potential reason that most traditional KGC models involve encoders and training

3 Literature Review

is that decoders are not as popular and powerful as today, like GPT-4 ([OpenAI, 2024a](#)).

3.3.2 Predefined Types and Predicates

The traditional KGC approach often extracts named entities and relations that fall within predefined sets. However, documents are usually diverse and contain named entities that are not limited to the predefined type set or relations that fall outside the predefined predicate set. In this case, the built KG may not be able to capture the information in the original document to a large extent.

The potential reason is that (1) the use of LLM encoders often involves training, and back in the old days when LLM decoders were not powerful, encoders were still popular. Since examples must have a predefined finite number of entity types. After learning from these examples, the classifier may only regard the tokens that belong to the predefined type provided in the example dataset as named entities and ignore named entities that are potentially out of the predefined types, similarly for relation extraction tasks. (2) Even with the popularity of LLM decoders, some studies still tend to follow the convention limiting the types and relations when using decoders for KGC instead of fully utilising the capability of LLM decoders ([Xie et al., 2023](#); [Ashok and Lipton, 2023](#); [Wang et al., 2023](#)). (3) The convention of building a data model may also be the reason for having predefined types and predicates before KGC. Traditional data models like the relational model must have a schema predefined before storing the information like named entities and relations as mentioned in Section [2.2.3](#). However, KG is much more flexible and the storing of named entities before having a complete schema, like their types.

3.3.3 Multi-Stage KGC

The traditional KGC approach involves multiple stages. However, this can suffer from error propagation ([Papaluca et al., 2023](#)). The potential reason is that the use of an LLM encoder may not be able to achieve multiple tasks at once, such as named entity extraction and relation extract, because each stage may need to fine-tune the LLM to perform a specific task. LLM decoder, on the other hand, can be considered a question-and-answer chatbot. Even though a task may involve multiple stages, it can still generate some answers.

3.3.4 Relations Between Named Entities Only

Knowledge in documents is not just the relationship between explicitly named entities. For example, the text: **I**t is renowned for its contributions to research and education has no explicitly named entity mentions, but it conveys the relationship between “ANU” and the concept “contribution to research and education”. In other cases, a sentence can have zero named entities, e.g. **C**ats eat fish, but the text indeed conveys knowledge, i.e., the relation between the general term cat and fish. This reason for this is also similar to [3.3.2](#), where the use of training dataset can limit to LLMs to extract more relations.

3.4 Related Work on Addressing Traditional KGC

3.3.5 Remarks

Although traditional KGC contains several limitations. Many studies following traditional KGC can still be valuable to refer to. Many techniques in literature that apply traditional KGC can still be useful and be migrated to some new KGC approach. In fact, during the development of our pipeline, we have also referred a lot to the studies about traditional KGC.

Based on our literature, some of the more recent KGC approach can be divided into three main categories

Schema at the beginning but automatic No fixed Schema Schema at the end

3.4 Related Work on Addressing Traditional KGC

There have been some more recent studies that aim to tackle the limitations of traditional KGC. These studies both have some common features: (1) the use of LLM decoder and (2) the concept of Open Information Extraction, i.e., the entities and predicates extracted are not confined to some predefined set, leveraging the generative ability of LLM decoders.

3.4.1 Examples

[Papaluca et al. \(2023\)](#) developed an end-to-end KGC pipeline that combines the Named Entity Recognition and Relation Extraction step into a triple extraction task. This means Named Entities and Predicates are no longer extracted and separated but together as a triple, which can then directly be used to build nodes and edges in a KG. The pipeline does not limit the types of Named Entities being extracted. It is also not limited to Named Entity. As long as an entity is a subject or object or some relation formed by a sentence, they are extracted. The pipeline also utilises few-shots in-context learning, which withdraws relevant examples (sentence-triples mapping) from a knowledge base that is similar to the input sentence so that the LLM decoder can learn during the inference stage how a triple extraction looks like through the examples in order to provide high-quality result.

[Wei et al. \(2024\)](#) 's study automated the schema definition step in traditional KGC. For each sentence, their model first prompts an LLM of the potential types and relations within a sentence. During the Named Entity Recognition and Relation Extraction stage, the types and predicates generated by the previous LLM are used to extract the actual named entities and relations. By doing so, the named entities and relations extracted are also not limited. [Ding et al. \(2024\)](#) 's study also exhibits a similar approach that first prompts the LLM to construct the schema, including the entity ontology and relation ontology, using LLM dynamically. This scheme is constructed based on the document instead of confining it to some predefined set.

3 Literature Review

([Zhang and Soh, 2024](#))’s study has developed an Extract-Define-Canonicalise framework. Like [Papaluca et al. \(2023\)](#)’s pipeline, the (Subject, predicate, object) triples are directly extracted from texts. Following that is a Schema Definition and Canonicalisation, which aims to merge similar predicates and types during open information extraction to reduce redundancy. This can be considered as the reverse approach to traditional KGC, where the schema of the KG is defined at the end.

3.4.2 Remarks

The recent studies on addressing traditional KGC have many merits, such as not limiting the types of entities extracted, the idea of putting Schema Definition at the end ([Zhang and Soh, 2024](#)) to allow open information extraction, or defined schema at the start in a automatic and dynmaic manner ([Wei et al., 2024; Ding et al., 2024](#)). At the same time, this work may also contain some limitations. For example, (1) all ([Papaluca et al., 2023; Wei et al., 2024; Zhang and Soh, 2024; Ding et al., 2024](#)) focus on extracting entities and relations at a sentence level. However, documents can have some relations spanning across sentences, paragraphs, or even sections. (2) These works tends to focus on one or several subtasks of KGC, rather than presenting a full KGC pipeline. For example, [Papaluca et al. \(2023\)](#) and [Wei et al. \(2024\)](#) focus on the information extraction stage of KGC. However, Coreference Resolution and Entity Disambiguation are also essential during KGC. With them, the output KG can be noisy and contain redundant nodes and relations. Therefore, it is necessary to have a more complete KGC pipeline spanning all the necessary stages (3) These works are mainly developed based on proprietary and large-scale LLMs like GPT-4, which can potentially lead to more carbon emissions and costs ([Papaluca et al., 2023; Wei et al., 2024; Zhang and Soh, 2024; Ding et al., 2024](#)). The closed property and the reliance on API and network connection also make it hard to measure some performance metrics, especially the runtime.

Chapter 4

Design of the Local Holistic KGC Approach

Based on the framework of traditional KGC, the ideas from more recent non-traditional KGC and all of their limitations found, we have proposed our local KGC from the documents approach; we would refer to it as the holistic KGC approach as we aim to extract more information from documents than the traditional none.

4.1 Demonstration of the Local Holistic KGC Approach

Our local holistic KGC approach aims to construct a local KG from text documents that capture *any relationship* between entities of *any type*. Entities involve anything that can syntactically appear as some mentioned in the original text, no matter if it is a Named Entity, like “ANU” or a concept, like “University”. The mention can be explicit, like “Amy”, or implicit, like “It”.

Our proposed holistic KGC process is divided into the following steps.

1. Entity Extraction, Typing, and Classification (Figure 4.1).
2. Entity Disambiguation and Coreference Resolution (Figure 4.1).
3. Relation Extraction (Figure 4.2).
4. Schema Generation (Figure 4.2).

Compared to the traditional one, the key difference is that schema definition is removed and replaced by schema generation at the end. This is because we are going to build a KG for free without aligning it with any schema. The graph in Figure 4.2 also represents the output KG. Also, the output KG from the same text source using the traditional KGC process is shown in Figure 4.3 for comparison.

4 Design of the Local Holistic KGC Approach

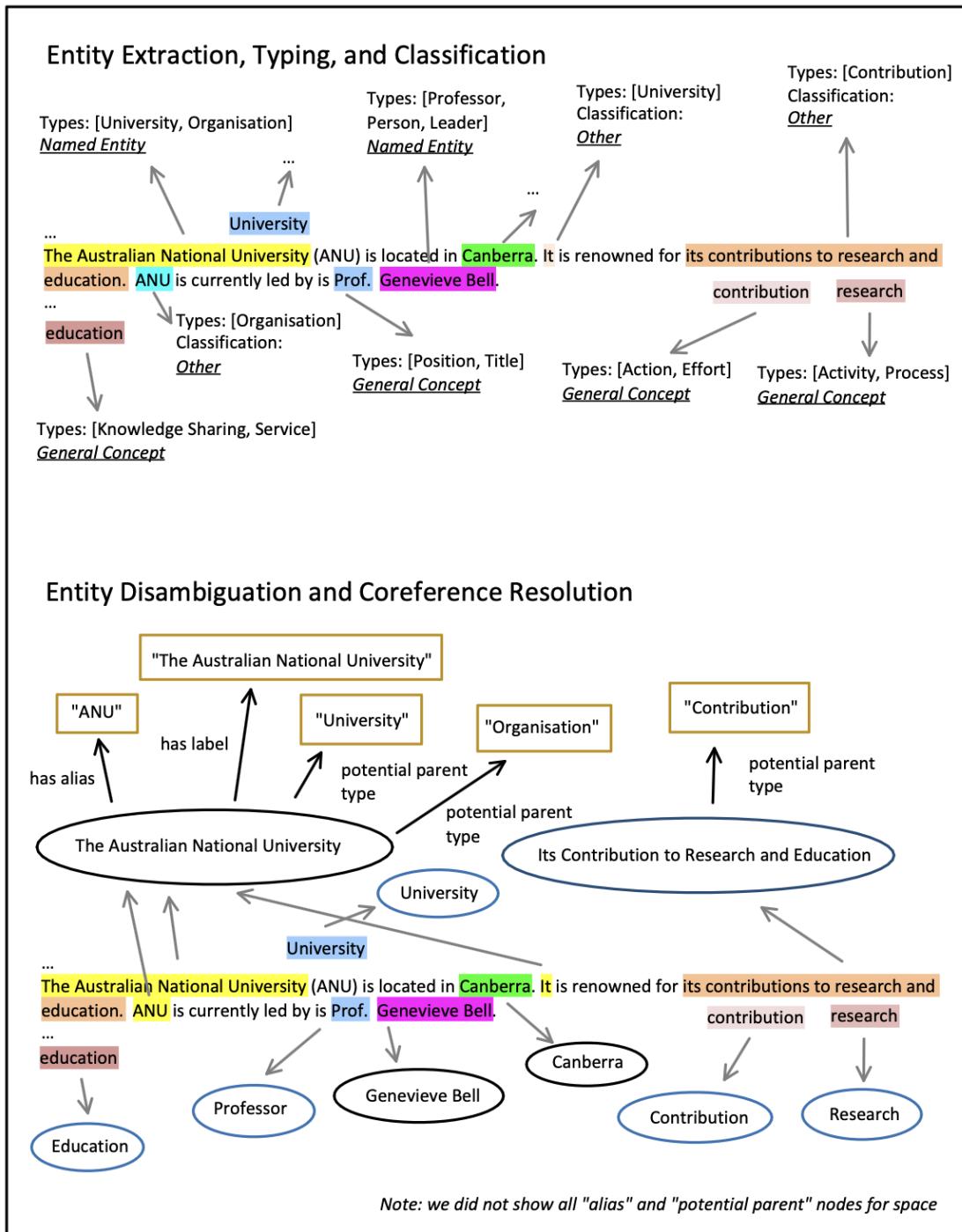


Figure 4.1: Holistic KGC, Steps 1-2

4.1 Demonstration of the Local Holistic KGC Approach

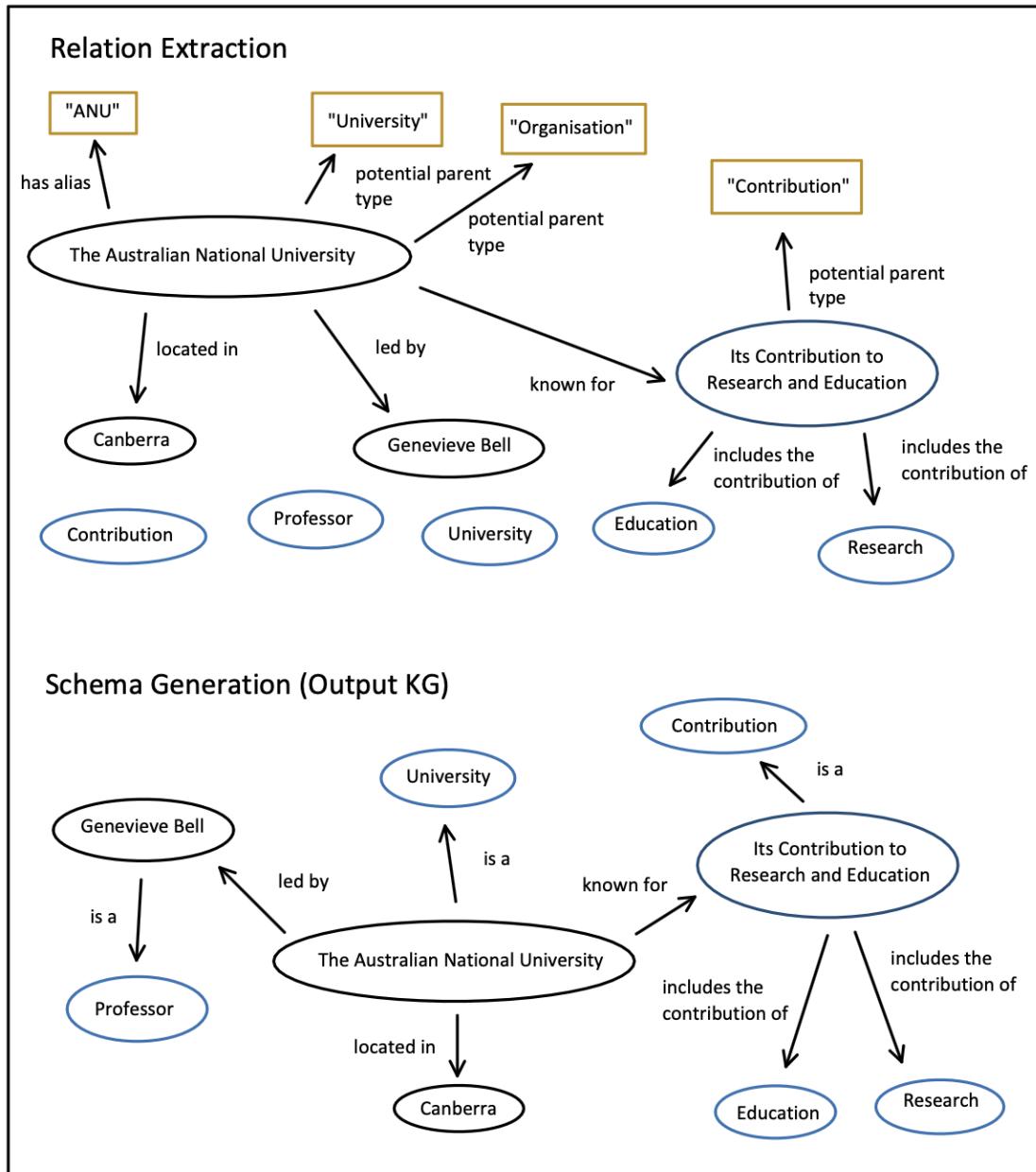


Figure 4.2: Holistic KGC, Steps 3-4

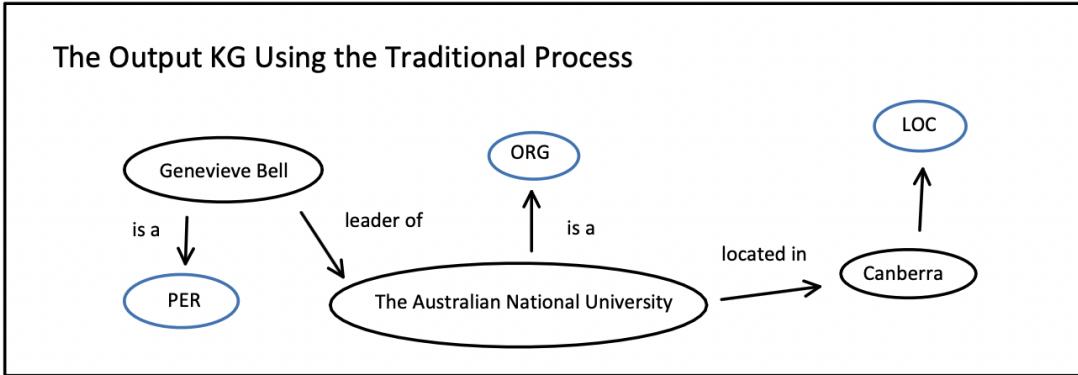


Figure 4.3: The output KG from the same text source using the traditional KGC process

4.1.1 Entity Extraction, Typing, and Classification

Instead of extracting mentions to Named Entities only as what the traditional KGC approach suggests (Hogan et al., 2020), our approach aims to extract all entities. However, we want to have a clear classification for the entities extracted. In this case, when inferring our constructed graph, people can filter only the Named Entities, for example. The classes we have set are Named Entity, General Concept, and Other.

- Named Entity involves any unique, specific real-world object (Hogan et al., 2020), such as “ANU”.
- General Concept involves any basic or non-composite concept, such as “University”, typically can be found in a dictionary or encyclopedia. A Concept or entity type is an abstract thing in the real world that represents a class of Named Entities with shared attributes (Elmasri and Navathe, 2015).
- Other involves anything else considered as an entity but falls outside the two above, which can include but is not limited to:
 - Non-General Concept: The concept that is not general, It can sometimes be made up of several general concepts, such as “tool that maps JSON into its RDF Graph representation.” representing any JSON-to-RDF mapping tools, but there may not be a general term in a dictionary or encyclopedia to describe it.
 - Any pronoun or referential expression, that implicitly refers to some specific thing, e.g., “Its Contributions to Research and Education,” indicating that this only refers specifically to ANU’s contributions rather than a general concept. This entity may link to another part of the document where ANU’s “Contributions to Research and Education” are discussed in detail. Unlike the Named Entity “ANU”, it may not have a proper name.

4.1 Demonstration of the Local Holistic KGC Approach

- Undefined or ambiguous entity: an entity that has not been fully resolved as a Named Entity or General Concept, such as the Entity represented by the term “It” a document, where the specific Named Entity or Concept the term “It” refers to, such as “ANU” or “Knowledge Graph”, has not been or cannot be identified.
- Most attributes of an Entity syntactically appear in the original document except the label, aliases, description and potential types. For example, “1946” and “u1234567”. We treat them as entities instead of literal because of the reason discussed in [2.3.2](#). We want to model both (“ANU”, “founded in”, “1946”) and (“1946”, “is a”, “special”). We do not treat “1946” as a literal because, in RDF-KG, all subjects must be an entity, i.e., identified by an IRI or local identifier. In contrast, as previously shown in [Figure 1.6](#) in [Section 1.4](#) and in [Figure 4.2](#), the label, aliases, and description of an Entity will remain represented as Literal Nodes, i.e., a square box in our figures.

It can be easily noticed in [Figure 4.1](#) that we aim to extract almost all nouns at different granular levels in the text. It may lead to a discussion as to whether to extract **its contributions to research and education** as a whole or two separate mentions. However, in our research, we have decided to keep things simple and make sure every mention extracted must be exactly presented in the original text.

There is also a long-term discussion in the Natural Language Processing field on whether to extract separately the possessive terms like “its” as mentions ([Xue et al., 2010](#)) and resolve it during the Coreference Resolution. However, we keep things simple and treat the whole term “Its Contributions to Research and Education” as some specific thing. During Coreference Resolution, we will not resolve the term “It” dedicatedly inside ANU. However, if there is a phrase elsewhere in the document that appears as “ANU’s Contributions to Research and Education”. In this case, we will then merge the two expressions into a single “ANU’s Contributions to Research and Education”.

Note that in some KGs, such as Wikidata, may not clearly distinguish between Named Entities and Concepts ([Wikidata, 2024](#)). For example, the Entity Knowledge Graph can be a Named Entity in the paper that first introduces it, but it becomes a Concept over time. However, we will maintain such a clear distinction for simplicity because we are building each local KG from a single document.

4.1.2 Entity Disambiguation and Coreference Resolution

This step is most similar to the traditional KGC process. However, the task is done together with all named entities, general concepts, and other classes instead of named entities alone. In addition, if we have successfully merged a “Named Entity” node, e.g., ANU, with a “Other” node, e.g., It. We can then essentially put some element out from the “Other” class to the more specific classes, “Named Entities”, and “General Concept”.

4 Design of the Local Holistic KGC Approach

4.1.3 Relation Extraction

This step is also similar. However, we aim to relate nodes regardless of their class. This is different from the traditional process, which usually focuses on relating named entity nodes. In addition, the predicate we are focusing on is everything other than the relationship like, `is a`, `subclass of`, and `instance of`, as these predicates define the ontology of the KG, telling what types of entities the KG will have, and we will resolve them in the next step. Our approach aim to extract not only relations within a local domain, like a single sentence, paragraph, or section, but also at the document level. Although it is not explicitly shown in the example above, it will be shown in our implementation section (Sections 5 and 9).

4.1.4 Schema Generation

This step is to build up the ontology of the KG. This is because, in the traditional KG process, the ontology, which is part of the KG schema, is usually predefined manually at the beginning before the actual automatic KGC (Hogan et al., 2020). Here, we will let machines design the schema for our KG based on the current nodes. The step of schema generation also aims to merge equivalent predicates with different names, such as `head of` and `is the header of`, just like **Coreference Resolution** to make our output KG more concise.

4.2 Comparison to Traditional KGC

Figure 4.3 and Figure 4.2 show the comparison of the output KG using the traditional KGC process and our proposed holistic KGC approach. It was found that

- More nodes are extracted from the original text other than named entities.
- More relations are extracted.
- The named entity nodes are assigned to more specific types instead of the list of predefined types.

Note that the relationship between `Canberra` and `Location` is missing in the KG output using our proposed method. This is because we only focus on using and linking everything that is **explicitly presented in the original text**. The main reason is that we are building local KG, and it is impossible to capture all human knowledge. For example, `Canberra - is a -> Capital City - is a -> City - is a -> Location`, etc. Of course, if there are the terms `Location` or `Capital City` appearing elsewhere in the text, we shall link them with `Canberra`.

4.3 Comparison to other Non-Traditional KGC Approach

Unlike (Papaluca et al., 2023) and (Zhang and Soh, 2024), which turn the information extraction process, i.e., the named entity recognition and relation extraction process in

4.3 Comparison to other Non-Traditional KGC Approach

traditional KGC, into a single step, we maintain them as two separate steps. Having two steps provides better manageability since LLMs are already a black box (Pan et al., 2023). It also enables the storage of intermediate results to have better recoverability from failure.

We will inherit the merits of all (Papaluca et al., 2023; Wei et al., 2024; Zhang and Soh, 2024; Ding et al., 2024)’s work that does not limit the types and predicates during the named entity recognition and relation extraction process.

Our extraction process will not be limited to Named Entities as traditional KGC, but also all the entities in the sentence, like a concept, similar to Papaluca et al. (2023)’s work. However, we will have a clear classification for an entity into either Named Entity, General Concept, or Other to create a more structured KG.

Similarly to Zhang and Soh (2024)’s work, our pipeline design will put the schema definition process back to the end as schema generation. That is, during the relation extraction process, we do not limit the predication extracted by the LLM. However, in the end, we will still normalise a list of predicates back to a limited set of predicates by merging similar predicates, e.g., “`founded in`” and “`established in`”, to reduce the redundancy during the open information extraction. In addition, the schema generation process will also create an ontology for our built KG, which creates better parent-children relations between all entities in our graph.

Unlike (Papaluca et al., 2023; Wei et al., 2024; Zhang and Soh, 2024; Ding et al., 2024)’s work, we maintain the process of Coreference Resolution during our KGC as in traditional KGC to reduce the redundant nodes.

Unlike (Papaluca et al., 2023; Wei et al., 2024; Zhang and Soh, 2024; Ding et al., 2024)’s work, during the actual implementation of our pipeline, we will also consider both local (sentence-level) relations and global (document-level) relations during the relation extraction process. By doing so, it also makes it hard for our work to become a single triple extraction process, like (Papaluca et al., 2023; Zhang and Soh, 2024)’s work because it can be hard to feed the whole document into the LLM decoder and ask it to extract triple. What we will do is first extract the entities, like in traditional KGC, and then filter the most important entities and then perform pair-wise comparisons to see if there are any relations exist within these entities, like what many traditional KGC work did (Wang et al., 2019; Tang et al., 2020; Wang et al., 2020).

We will also utilise the lightweight and open-source LLM during the development of our pipeline, a 4-bit quantisation version of LLaMA 3 7B, which is only about 4 GB and runnable in many household computers. However, using a light-weight LLM to develop our pipeline also brings challenges because the LLM is considered less “smarter” and will exhibit the common issues of LLM decoder, like the hallucination issue as discussed in Section 2.4.2, to a larger extent,

Essentially, our work will lead to another alternative to traditional KGC. There may not be a definition answer to which one is better. We are just together discovering different

4 Design of the Local Holistic KGC Approach

paths to address the limitations of traditional KGC.

4.4 SWOT Analysis for Our Holistic KGC Approach

The following is a summary of some strengths, weaknesses, opportunities, and threats about our holistic KGC approach.

4.4.1 Strengths

Can extract any entities and build their relation as long they syntactically appear as some terms in the document. Either an explicit term like ANU or an implicit term like it. This is included but not limited to:

Named Entity - Named Entity: (ANU, located in, Canberra)

Named Entity - Concept:
(ANU, known for, Contribution to research and education)

Named Entity - Attribute: (ANU, founded in, 1946)

Named Entity - Event: (Ben, attends, 2024 S2 ANU's graduation ceremony)

Can extract relations that span across sentences, although not demonstrated in the example, which will be shown in Section 5.

4.4.2 Weaknesses

(1) It cannot extract entities that do not appear as a term in the document but can be derived based on the semantics. For example, consider the example shown before:

"Amy was practising piano at 5:00 PM"

The entity that does not syntactically appear in the sentence is the specific moment when Amy is practising piano. In KGC or broader NLP, this is also referred to as Event, ([Zhong et al., 2023](#)). We will simply call it as

moment_1.

Since a document may consist of multiple moments, we use 1 for disambiguation.

Therefore, in the RDF graph, ideally, the relations that can be extracted are:

(moment_1, has time, 5:00 PM)
(moment_1, involves the subject, Amy)
(moment_1, involves the object, Piano)

However, in our constructed KG, it will only be:

(Amy, practises, Piano)

4.4 SWOT Analysis for Our Holistic KGC Approach

which tells the general fact only, and the temporal information is lost.

However, if the original sentence is written as

Amy has a piano practising session starting at 5:00 PM.

Then, our KGC approach should be able to represent the sentence losslessly as:

(Amy, has, Piano practising session_1)

(Piano practising session_1, starts at, 5:00 PM)

This is because the term piano practising session syntactically appears in the original sentence.

The reason we left this weakness is not only because of the time and difficulty of capturing events but also because of the limitation of the RDF graph, where extra nodes often need to be created to capture temporal information. This may lead to future work using more expressive graphs, like the property graph and the RDF* graph, to capture more temporal information from original documents.

(2) The second weakness is that our KGC approach still contains multiple stages. This is almost unavoidable because the stages, like Coreference Resolution, depend on the extraction stage. However, this can lead to error propagation.

4.4.3 Opportunities

The open information extraction during the holistic KGC process allows the preservation of the original information in the document to a larger extent. This is important if we target a more complete knowledge representation of the original document.

4.4.4 Threats

The open information extraction may reduce the conciseness of the output KG, especially if we only want to extract the key entities and the relationships between them. This may increase the complexity when inferring the graph to extract information during its application.

The open information extraction process may rely on an LLM decoder instead of a decoder since a decoder can better extract entities and relations that are not limited to predefined types. However, decoders internally have a higher complexity than decoders, being widely used in traditional KGC, (Pan et al., 2023). Therefore, although it may not require many computational resources during training rather than preparing the prompts, during the running of the pipeline, i.e. the inference stage, it may consume much higher computational resources compared to those KGC pipelines based on the traditional KGC approach built upon LLM encoders, as previously discussed in Section 2.4.5.

4 Design of the Local Holistic KGC Approach

The use of decoders may also suffer from its hallucination issue, where entities and relations that do not exist in the document are imagined and extracted by the LLM, which further decreases the preciseness, as previously discussed in Section [2.4.2](#).

Chapter 5

Overview of the Implementation of the Local Holistic KGC Pipeline

The main outcome of this research is a functional KGC pipeline prototype that is capable of converting a semi-structured academic paper into its KG representation. Therefore, in this section and the following Sections 6, 7, 8, 9 10, we will take a close look at the pipeline and discuss the working principle behind it. Meanwhile, we will also discuss the key decisions made during the implementation of the pipeline.

5.1 Interface

- **Input:** The semi-structured representation of an academic paper (RDF-Based Metadata Graph) in TTL or our customised JSON format.
- **Output:** The Knowledge Graph representation of the paper (RDF-Based Knowledge Graph) in TTL or our customised JSON format.
- **Language:** Python 3.12.4.
- **Software Platform:** Linux, MacOS.
- **Hardware Platform:** CUDA, Metal.
- **LLM Used:** Meta-Llama-3-8B-Instruct-Q4-0, bge-english-base, bge-m3.
- **Key Packages Used:** GPT4ALL (loading LLM decoders), FlagEmbedding (loading LLM encoders), NLTK (providing basic natural language processing functionality like lemmatisation).
- **Availability:** <https://gitlab.anu.edu.au/u1085404/anu-scholarly-kg/-/tree/master/llm-based/LLM-KGC-v0>

5 Overview of the Implementation of the Local Holistic KGC Pipeline

The summary of the interface is shown as above. Note that:

- GPT4ALL (GPT for All) and GPT-4 are two distinct things. GPT4ALL provides a platform to load many LLM decoders into Python. It also enables the Chat Completion functionality of a decoder model ([Nomic, 2024](#)), i.e., the decoder model loaded by GPT4ALL will answer a user's prompt, instead of generating anything that continues the prompt, as previously discussed in Section [2.4.4](#).
- Although RDF-Graph is commonly stored in the format of TTL, as discussed in Section [2.2.2](#), we have chosen to use JSON to store the KG for some special reasons. In fact, our pipeline will *first convert a TTL-format KG into our customised JSON-format KG* and work on this customised JSON format through the KGC stages because a JSON file can be loaded into Python as a Python Dictionary. In contrast, the access of TTL files relies on SPARQL Query, which is not natively supported by Python. More will be discussed in the next Section [5.4](#).

5.1.1 LLM Encoders and Decoders Used

Decoder 1:

- **Named:** Meta-Llama-3-8B-Instruct.Q4_0.gguf
- **Size:** 4.66GB
- **Context Limit:** 8192
- **Availability:** <https://huggingface.co/QuantFactory/Meta-Llama-3-8B-Instruct-GGUF/tree/main>

Encoder 1:

- **Named:** BAAI--bge-base-en-v1.5
- **Size:** 438.9MB
- **Context Limit:** 512
- **Availability:** <https://huggingface.co/BAAI/bge-base-en>

Encoder 2:

- **Named:** BAAI--bge-m3
- **Size:** 2.29GB
- **Context Limit:** 8192
- **Availability:** <https://huggingface.co/BAAI/bge-m3>

In the later sections and chapter, especially when presenting an algorithm, we will refer to them as LLaMA, BGE, and M3, respectively.

5.1.2 Development Set Used

Paper: “J2RM: an Ontology-based JSON-to-RDF Mapping Tool” (Rodríguez Méndez et al., 2020) Availability: <https://ceur-ws.org/Vol-2721/paper593.pdf>

We have chosen a single academic paper from ASKG as our primary development set (validation set). Although, there is only one paper. The paper itself consists of a number of sections, paragraphs, and sentences, which is enough to cover relatively diverse linguistic patterns and help us observe different edge cases during our NLP tasks. It can also accelerate our development, but the drawback is that it can still potentially make the implementation of our pipeline biased to the specific subset of linguistic patterns contained in the paper.

We have chosen SciERC (Luan et al., 2018) and DocRED (Yao et al., 2019) as our secondary datasets. These datasets both provide mappings between documents and KGs. However, neither is considered to contain a full article. SciERC’s sources come from academic papers, but every single sample contains the abstract only. DocRED’s sources mainly come from Wikipedia, but they are also shortened, containing only a single paragraph. Therefore, the high performance of these datasets does not necessarily mean that our pipeline performs well.

Furthermore, the ground truth KGs from these two datasets are considered as KGs based on traditional KGC according to our literature review. This is because the target KGs of both datasets only consist of a limited number of predefined entity types and predicates. For example, SciREC only contains predefined six academic entity types, like Task and Method, and seven academic predicates, like Use-for. DocRDF contains six general entity types, like Person and Location and 96 predicates. Therefore, if we only aim to achieve a high benchmark score on these two datasets, we essentially confine our pipeline to produce traditional KG, instead of the KG based on the idea of open information extraction. However, they are still useful in some specific cases, and we have used them during our development.

Both our primary and secondary datasets will be mentioned in Section 6 to Section 10 to demonstrate some decisions made during the development of our pipeline, especially our prompts. This can be compared to hyper-parameter turning in traditional machine learning, but we are turning the prompts in this study.

Next, we will go through both the high-level data structure and low-level data structure of our input and output. This is highly relevant to our algorithms, which will be discussed in the later sections.

5.1.3 High-Level Input and Output Data Structure

The high-level input and output are shown in 5.1. This is slightly more detailed than the one shown in the introduction. It is shown here for comparison with other low-level inputs and outputs. Note that we have eliminated some intermediate fields stored in

5 Overview of the Implementation of the Local Holistic KGC Pipeline

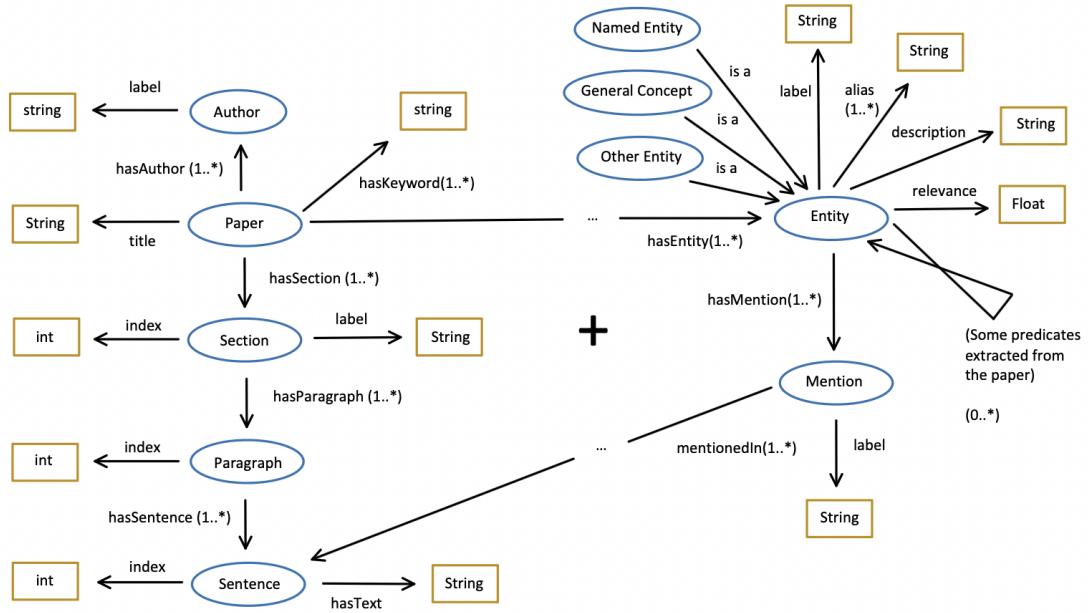


Figure 5.1: High-Level Input and Output

this data structure. They are important during the processing but are deleted when the pipeline finishes.

5.1.4 Low-Level Input (TTL and JSON)

TTL Pseudo-Code

```

paper_1_iri      is_a          Paper ;
                  has_title     "Research on the History of ANU" ;
                  has_author    "Amy",
                  has_author    "Ben" ;
                  has_keyword   "ANU",
                  has_keyword   "History";
                  has_section   section_1_iri,
                  has_section   section_2_iri,
                  has_section   section_3_iri .
...

section_2_iri    is_a          Section ;
                  has_index     2 ;
                  has_label     "Introduction" ;
                  has_paragraph paragraph_1_iri,
                  has_paragraph paragraph_2_iri,
                  has_paragraph paragraph_3_iri .

paragraph_1_iri  is_a          Paragraph ;
                  has_index     1 ;
                  has_sentence  sentence_1_iri,

```

```

        sentence_2_iri,
        sentence_3_iri .

...
sentence_1_iri      is_a          Sentence ;
                    has_index    1 ;
                    has_text     "ANU is a university located in Canberra.".
...

```

JSON Pseudo-Code

```

{
  "iri": "paper_1_iri",
  "title": "Research on the History of ANU",
  "authors": [
    "Amy",
    "Ben"
  ],
  "keywords": [
    "ANU",
    "History"
  ],
  "sections": [
    {...},
    {
      "iri": "section_2_iri",
      "label": "Introduction",
      "paragraphs": [
        {
          "iri": "paragraph_1_iri",
          "sentences": [
            {
              "iri": "sentence_1_iri",
              "text": "ANU is a university located in Canberra."
            },
            {
              "iri": "sentence_2_iri",
              "text": "It has a long history."
            },
            {...}
          ]
        },
        {...},
        {...}
      ]
    },
    {...}
  ]
}

```

Up to now, it should be clear why we prefer JSON to TTL. The original graph data structure, RDF-Graph, has no sense of order and hierarchy. Everything is just a triple sitting on a plane. The hierarchy has to be inferred based on the relations of nodes, and the orders of sections, paragraphs, and sentences have to be stored in extra index nodes. However, the semantics of the semi-structured representation of a paper is hierarchical and ordered.

5 Overview of the Implementation of the Local Holistic KGC Pipeline

In contrast, JSON is hierarchical, and its list is ordered. When JSON is implemented by Python (> 3.7), i.e., it is parsed by Python into Python Dictionary or a Python Dictionary is dumped into JSON, *its key is also ordered*. That is, `iri` and `title` are stored in a list instead of a set. This is why JSON (disk-level) or Python Dictionary (memory-level) is chosen. Note that an RDF-Graph or RDF-KG is a high-level concept. How such a graph is stored at a low level can vary.

5.1.5 Low-Level Output (TTL and JSON)

TTL Pseudo-Code

```
/* All triples in the input are preserved plus: */

entity_1_iri      is_a          Named_Entity ;
                   has_label      "The Australian National University" ;
                   has_alias      "ANU",
                   has_description      "The Australian National University",
                   has_relevance_score  "Australian National University" ;
                   has_mention     "...";
                   predicate_1_iri   entity_2_iri;

entity_2_iri      is_a          Named_Entity ;
                   has_label      "Canberra" ;
                   has_alias      "Canberra",
                   has_description      "CBR",
                   has_relevance_score  "...";
                   has_mention     entity_2_mention_1_iri,
                   ... .

entity_1_mention_1_iri  is_a          Mention ;
                        has_label      "ANU" ;
                        mentioned_in    sentence_1_iri .

entity_2_mention_1_iri  is_a          Mention ;
                        has_label      "Canberra" ;
                        mentioned_in    sentence_1_iri .

entity_1_mention_1_iri  is_a          Mention ;
                        has_label      "ANU" ;
                        mentioned_in    sentence_1_iri .

predicate_1_iri      is_a          Predicate ;
                   has_label      "located in" ;
                   has_description  "...".

Named_Entity         sub_class_of  Entity .
General_Concept       sub_class_of  Entity .
Other_Entity          sub_class_of  Entity .
```

JSON Pseudo-Code

```
{
  /* All fields in the input are preserved plus */
  "nodes":{
```

```

"entity_1_iri": {
    "label": "The Australian National University",
    "aliases": [
        "ANU",
        "The Australian National University",
        "Australian National University"
    ],
    "node_type": "Named Entity",
    "description": "...",
    "relevance_score": 0.9,
    "mentions": {
        "entity_1_mention_1_iri": {
            "local_name": "ANU",
            "reference": "sentence_1_iri"
        },
        ...
    }
},
"entity_2_iri": {
    "label": "Canberra",
    "aliases": [
        "Canberra",
        "CBR"
    ],
    "node_type": "Named Entity",
    "description": "...",
    "relevance_score": 0.8,
    "mentions": {
        "entity_2_mention_1_iri": {
            "local_name": "Canberra",
            "reference": "sentence_1_iri"
        },
        ...
    }
},
...
},
"edges": {
    "predicate_1_iri": {
        "label": "located in",
        "description": "..."
    },
    ...
},
"triples": [
    [
        "entity_1_iri",
        "predicate_1_iri",
        "entity_2_iri"
    ],
    ...
]
}

```

The key difference is that, instead of storing everything in the KG as triples, like TTL. We have stored it using a hybrid approach.

- For the relations that only belong to an entity, e.g., aliases, descriptions, and mentions, they are stored directly under the Entity's local fields instead of the global field "triples".

5 Overview of the Implementation of the Local Holistic KGC Pipeline

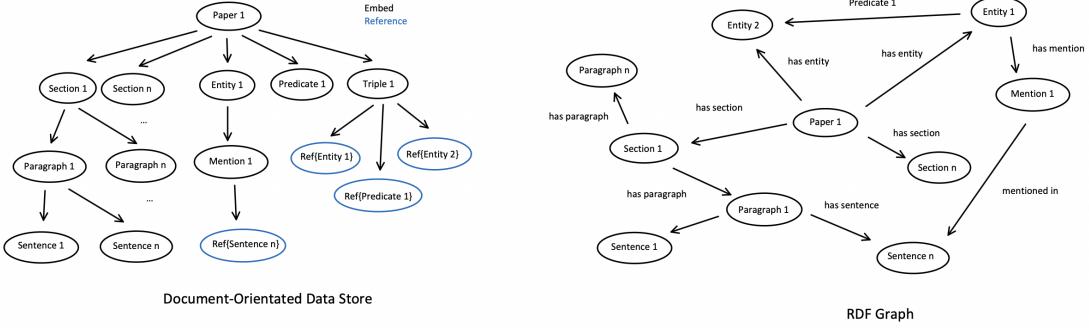


Figure 5.2: Original Input and Modified Input

- For the relations between any two entities, they are stored in the global fields of the fields, like like in TTL. Therefore, our overall KG in JSON is stored as a list of nodes, plus a list of edges, plus a list of triples.

In this way, we can easily observe any local relations about the entities, e.g, their labels, and aliases, by checking the entity itself and any global relations, i.e., the relations between any two entities, in a single `triples` list. This makes it easy for us to directly observe and debug the relations between entities in the raw JSON file without the assistance of any KG viewing tool. Whereas in TTL,

In TTL, however, relations scatter among each entity to which the subject of the relation belongs, as shown in the TTL pseudo-code. In fact, in TTL, there is even no guarantee that the triple `(entity_1_iri, predicate_1_iri, entity_2_iri)` is under the entity `entity_1_iri`. It can be placed anywhere in the file, and there is a sense of ordering or hierarchy of triples. Therefore a graphical tool is usually needed to observe a KG in TTL format.

Our pipeline converts our intermediate JSON-base KG back to TLL as our output because, of course, TTL is the most common and widely supported way to store a KG.

Table 5.1 and Figure 5.2 provides a different view of the input and output of the pipeline. Essentially, our JSON-Based modified input can be regarded as a Document-Orientated Data Store, which maintains some hierarchical structure of the document as well as storing triples and representing semantics.

5.2 Pipeline Structure

Figures 5.3 and 5.4 collectively show the overall structure of our pipeline. The box coloured green indicates that this sub-stage involves an LLM decoder, the box coloured blue indicates LLM encoders are involved, and the box coloured black indicates the stage is purely algorithmic. The following sub-section will briefly go though each stage and

Table 5.1: Different View of Input and Output

Semantical	A Semi-Structured Paper	Intermediate KG	A KG of the Paper
Logical	RDF-Graph	Document-Orientated Data Store *	RDF-Graph
Disk	TTL	JSON	TTL
Memory	RDF Graph Object	Python Dictionary	RDF Graph Object

demonstrate how an input JSON/TTL Metadata Graph of a paper is converted into a JSON/TTL Knowledge Graph. The boxes placed in parallel means the substages are not dependent on each other and are ideally run in parallel.

5.2.1 Stage 0: Data Pre-Processing

The first stage involves converting TTL-Based KG into our customised JSON format so that when a KG is loaded into memory, it can become a natively-supported Python Dictionary for our later algorithm to process. Since our pipeline is primarily designed for taking semi-structured papers from ASKG as inputs, most of the time, our inputs are in TTL.

Another key sub-stage is the integrity and consistency check in order to prevent our later algorithms from failing. As mentioned in Section 2.2.3, the RDF Graph is designed to be highly flexible, and hence there are usually no constraints explicitly coded, unlike the relational model. For example, there is no guarantee that a Paper node has at least one section, even though we have stated such constraints in our ontology graph (Figure 5.1). In fact, when we first receive our input data from ASKG, there are indeed some integrity and consistency issues. For example, we have found that some papers have two titles and some papers have sentences or sections missing. These papers need rejection and reporting.

Therefore, this sub-stage checks the following:

- A paper has *exactly* one title.
- A paper has *at least* one author, keyword, and section.
- A section has *exactly* one subtitle.
- A section has *at least* one paragraph.
- A paragraph has *at least* one sentence.
- A sentence has *at least* one piece of text.

5.2.2 Stage 1: Entity Extraction, Typing, and Classification

As shown in Figure 5.3, this stage involves extracting all entity mentions in the text, assigning them one or more potential parent types, and classifying them into either

5 Overview of the Implementation of the Local Holistic KGC Pipeline

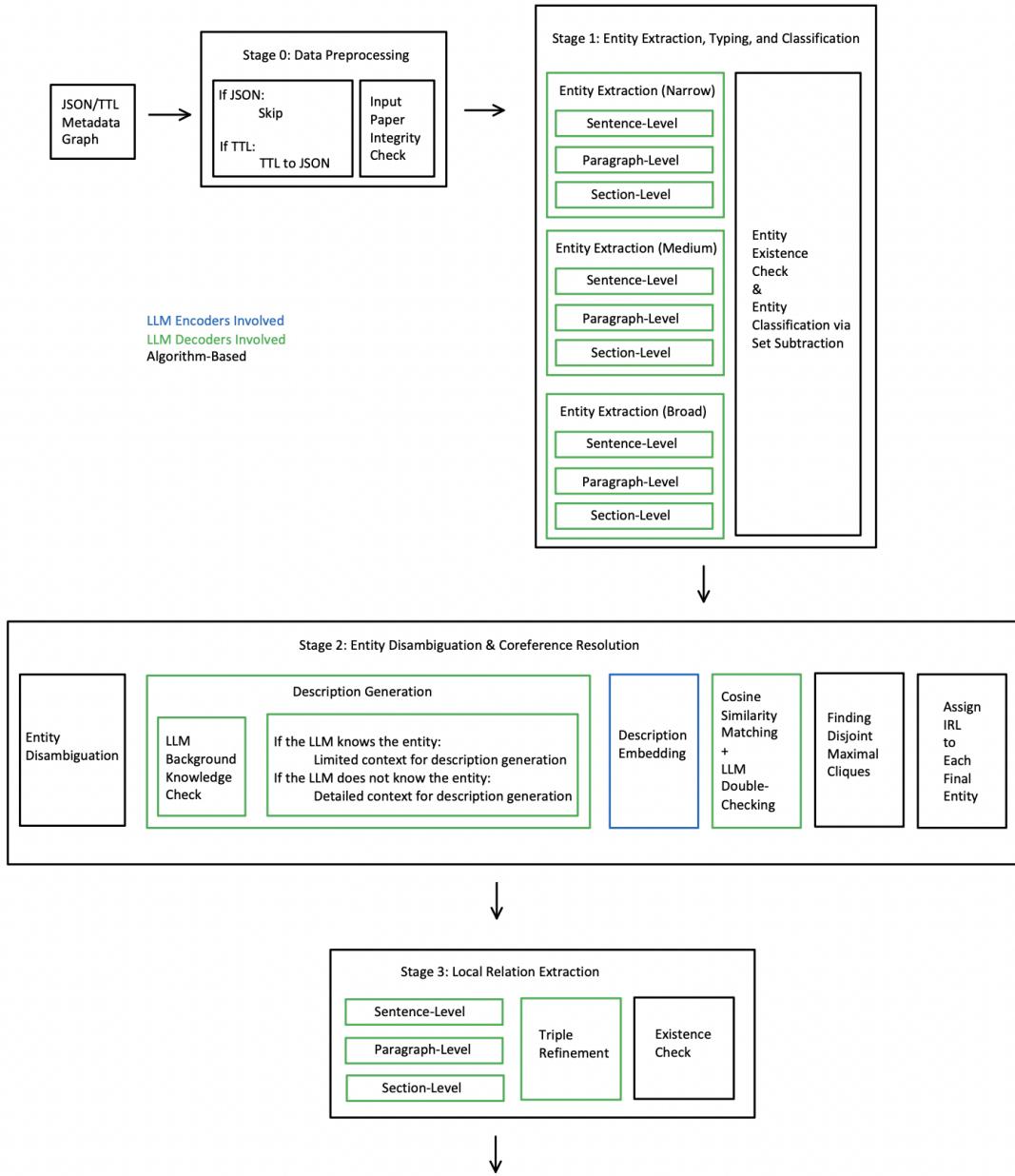


Figure 5.3: Pipeline Structure: Stage 0 - Stage 3

5.2 Pipeline Structure

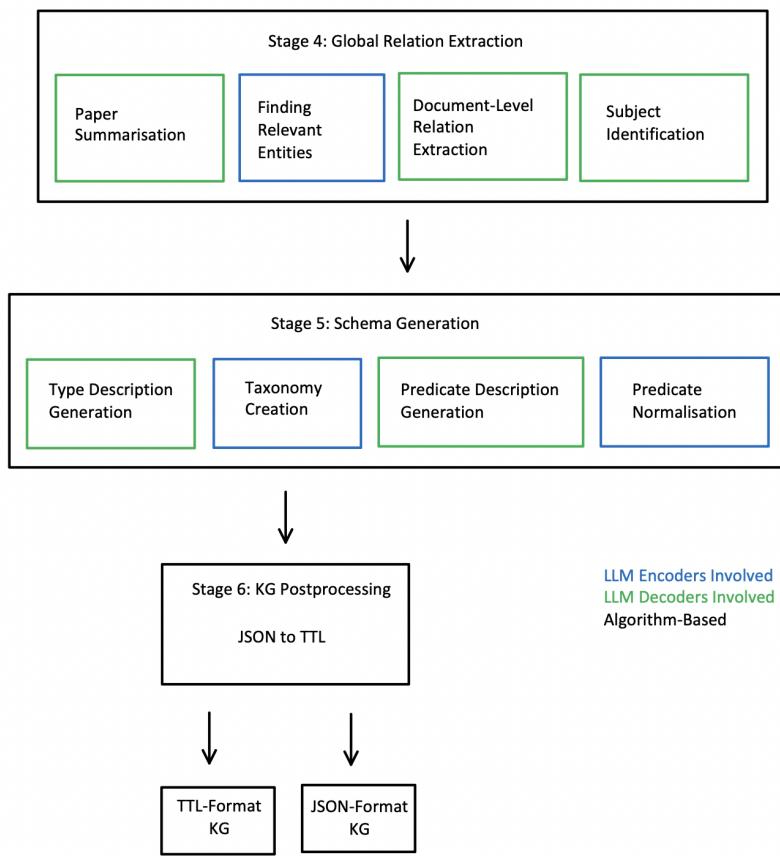


Figure 5.4: Pipeline Structure: Stage 3 - Stage 6

5 Overview of the Implementation of the Local Holistic KGC Pipeline

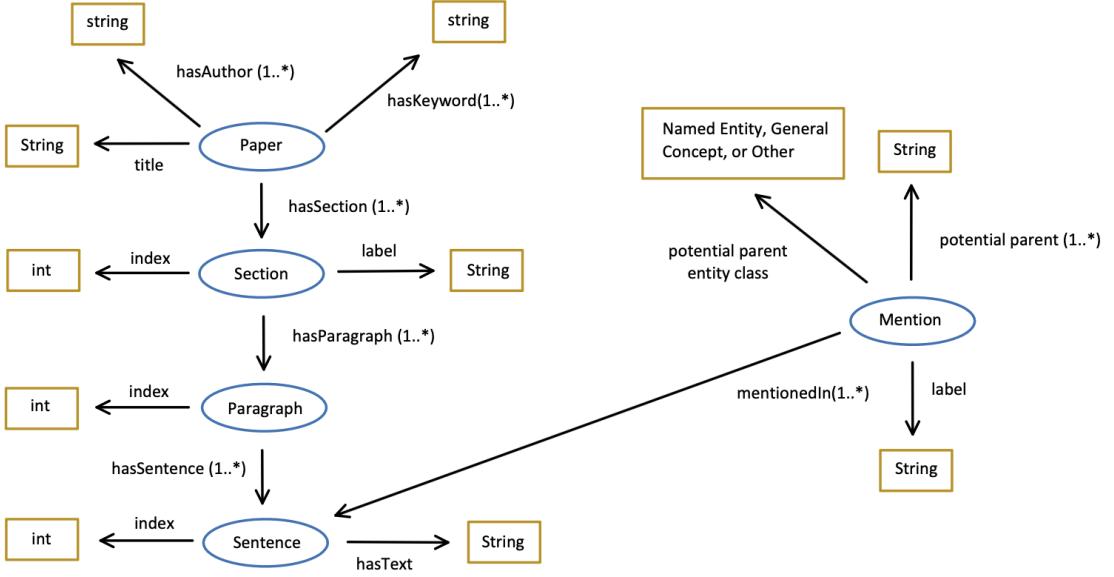


Figure 5.5: Intermediate KG Ontology after Stage 1

Named Entity, General Concept, or Other.

The extraction process can be divided into three sub-stages: narrow, medium, and broad. This is useful when we perform entity classification via set subtraction. The narrow entity extraction aims to extract only the named entity in the document. The medium entity extraction aims to extract both the named entity and the general concept. The broad entity extraction aims to extract all entities. Therefore, by set subtraction, we can classify our extracted entities into Named Entity, General Concept, and Other.

The extraction processes also go from sentence level (looping through every sentence) up to section level (looping through every section), i.e., fetching the LLM sentence by sentence or section by section to extract entities.

After the extraction, our pipeline checks if the entity is present in the document for each entity. The reasons for doing all these sub-steps will be discussed in Section 6.

After Stage 1, all Mention nodes will be created in our KG. Figure 5.5 shows the intermediate KG ontology after stage 1. Note that the entity nodes have not yet been created until Stage 2. Note that we use the term “type” to refer to a parent of an entity. For example, “ANU” has the type “University”. We use the term “class” to refer to our broad classification of entity, e.g., “ANU” has the class “Named Entity”.

5.2.3 Stage 2: Entity Disambiguation and Coreference Resolution

This stage, as shown in Figure 5.3, involves linking all mentions referring to the same thing together into Entity nodes. The ontology of the intermediate KG after this stage

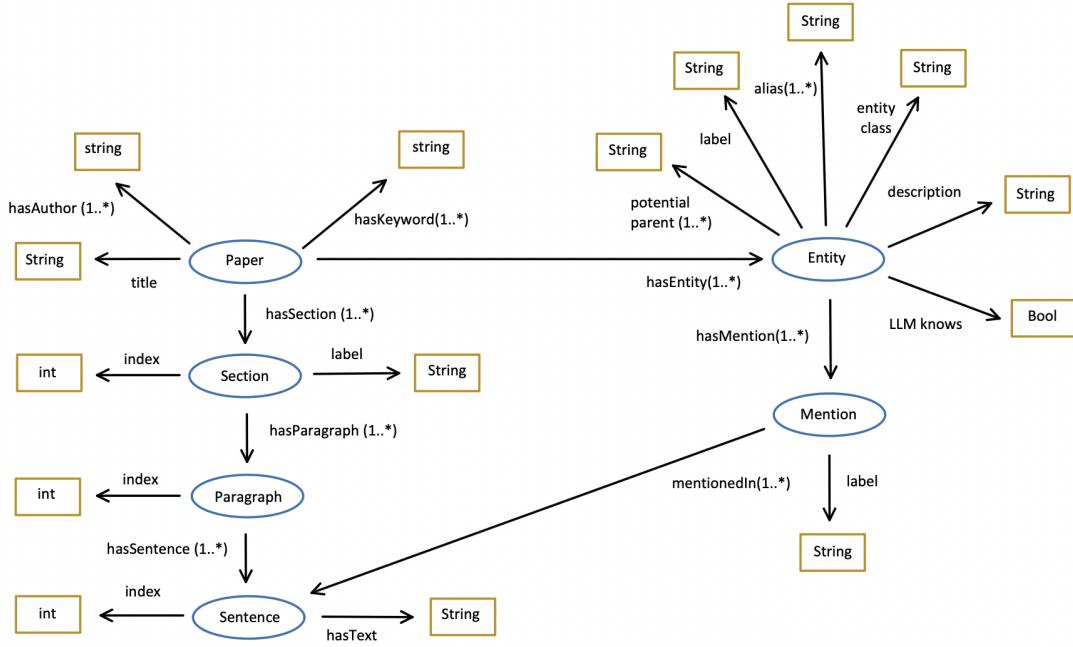


Figure 5.6: Intermediate KG Ontology after Stage 2

is shown in Figure 5.6. At the beginning of this stage, we assume that all Mentions refer to distinct entities. Therefore, we create an Entity node for each Mention.

We first perform Entity Disambiguation to merge entities with the same label (name).

Next, we prompt the LLM decoder to generate a description for each entity, whether how a description is generated is based on whether the LLM has background knowledge of this entity. This will ensure a higher-quality description.

Then, we use the description, the entity label, and the entity types to generate the embedding for each entity using the LLM encoder.

If it is found that two embeddings are similar, we prompt the LLM decoder to perform a double-check. If it is found that two entities are similar, we draw a temporary edge between these two entities.

We will merge a group of entities into a single entity only if they form a complete graph in our intermediate KG (cliques).

When merging any two entities into one entity, the mentions of the new entity are then the union of mentions of the two entities, similar to potential parent types and aliases. The label, i.e., the representative name of the entity, is chosen to be the label of the mention that first appears in the document, similar to the description. The other names that are not chosen are stored as aliases. This is also due to the assumption that

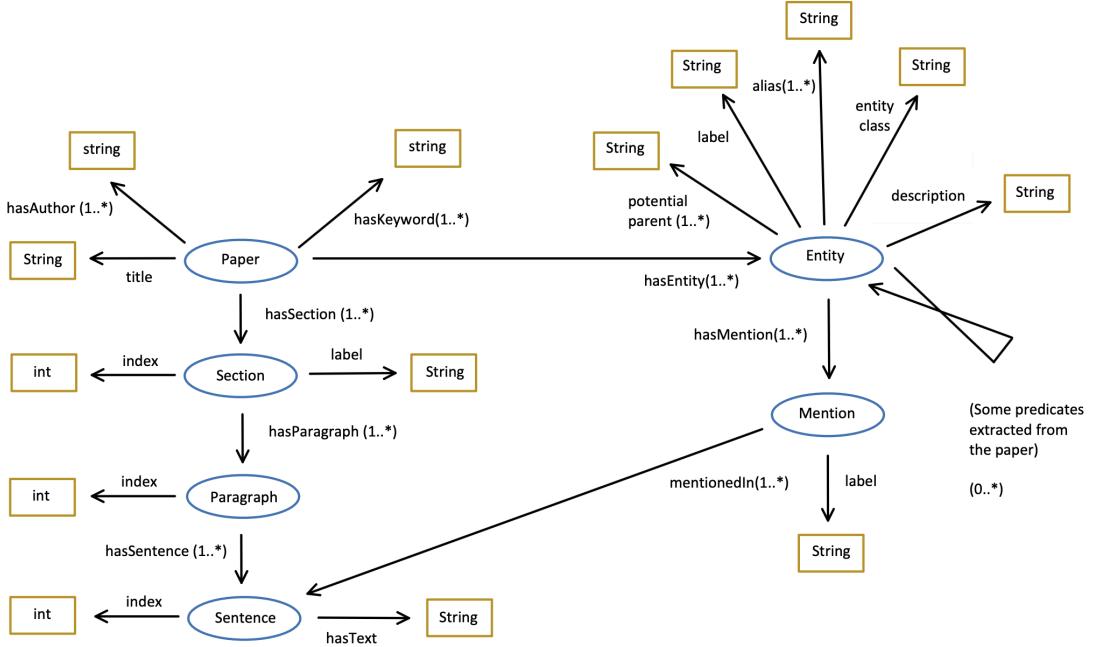


Figure 5.7: Intermediate KG Ontology after Stage 3

in academic writing, the entity that appears first is usually the full name, like “The Australian National University” instead of “ANU”, which is the most representative.

If an Other Entity is merged with a Named Entity or a General Concept, the resulting Entity will be a Named Entity or General Concept, respectively. If a General Concept is merged with a Named Entity, the resulting Entity will be a Named Entity.

After this stage, all Entity nodes in our KG are created and fixed with IRI assigned. The only things missing are the edges between Entities.

5.2.4 Stage 3: Local Relation Extraction

This stage, as shown in Figure 5.4, involves extracting the relations between entities within a limited context. For example, given that all the entities are found in a sentence, paragraph, or section, in the “(Subject, Predicate, Object)” form. we prompt the LLM decoder to extract the relations among them. During our development, we found that our lightweight LLaMA3 sometimes LLM puts predicate and object together and leaves the object slot blank. Therefore, for each triple with the object slot empty. We go through the extra triple refinement or triple decomposition step to fix these triples. Also, we check if both the subject and object of a triple exist in the document to prevent the hallucination issue of LLM, as discussed in Section 2.4.6.

The ontology of the intermediate KG after this stage is shown in Figure 5.7.

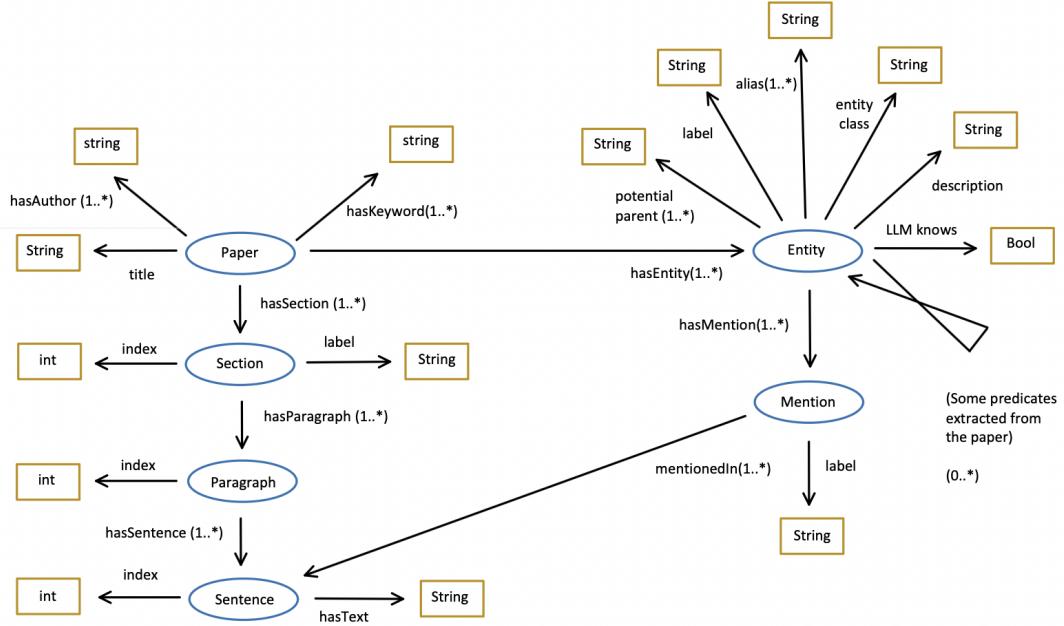


Figure 5.8: Intermediate KG Ontology after Stage 4

5.2.5 Stage 4: Global Relation Extraction

This stage, as shown in Figure 5.4, starts with shortening the original document until it fits the context limit of the LLM decoder. The shrinking is performed by prompting the LLM decoder to summarise one or more sections. The summarization is performed from the middle section of the paper up to the Abstract and Conclusion to ensure less information loss.

Then it calculates the relevancy score for each entity to the document by comparing the embedding of an entity and the embedding of a document.

Only 10% or up to 20 of the most relevant entities are selected for global relation extraction since this stage involves the complex pairwise comparison, which is $O(n^2)$. Then, for each entity pair, we fetch the LLM the shortened version of the document and extract a predicate among them.

During our development, we found that our LLM cannot effectively determine the subject and the object even if a predicate is returned. Therefore, we added another step. This step gives any two entities and their predicate extracted in the previous step to prompt the LLM decoder to determine the subject in this relation.

The ontology of the intermediate KG after this stage is shown in Figure 5.8.

5.2.6 Stage 5: Schema Generation

This stage can be divided into two parts: Taxonomy Generation and Predicate Normalisation.

Taxonomy Generation aims to create parent-child relations between entities in our graph if they have not been extracted during Stage 3 and Stage 4, such as linking “ANU” with “Public University” and “Public University” with “University”.

This step will utilise the potential parent generated in Stage 1. For all potential parents of an entity, like in Stage 2, we first generate a description and then embed it. After that, we compare these embeddings with the embeddings of the other Entity nodes in the graph. If a matching entity is found, we then say that the current entity is a child of the matching entity. This is essential for a more advanced entity typing step compared to one in Stage 1, where the types built are hierarchical instead of flattened.

Predicate normalisation is about merging similar predicates. As before, we first generate a description for each predicate and then merge them based on the similarity of their embedding. The ontology of the KG does not change after this step and maintains the one shown in Figure 5.8.

5.2.7 Stage 6: KG Post-Processing

This step is mainly about converting our customised JSON-format KG into TTL-format KG and deleting the intermediate fields, including the paper summary and potential parents. However, this is only the default option for the pipeline. Whether or not to keep these fields depends on the actual deployment. The ontology of the KG after stage 6 is the same as Figure 5.1.

5.3 Execution Diagram

The Python project structure of the pipeline is shown in Figure 5.9. It contains a master program “main.py” and a list of stage packages, where each stage package consists of a list of scripts. The execution diagram of the pipeline is shown in Figure 5.10. Since each stage has different dependencies, the master program spawns a process before running each stage to avoid too many packages loaded at once. After the process, a stage takes over control. It then imports only the packages needed during the current stage and runs each script in order. Each script first reads the result of the previous script stored on disk as “kg_x.json”. Then, it does its work and, finally, outputs another “kg_x.json” for the next script to read. In this way, all intermediate results are maintained, and if the pipeline fails, there is a recovery point. The drawback is that there is slightly more memory consumption. However, compared to tasks that involve calling LLMs, especially decoders, which can take minutes up to hours (will be shown in Section 11), the time spent on file input and output is negligible.

5.3 Execution Diagram

```
✓ LLM-KGC-DEMO
  ✓ data
    > input
    > output
  ✓ src
    ✓ stages
      > s0_kg_preprocessing
      > s1_entity_extraction_typing
      > s2_entity_resolution_disambiguation
      ✓ s3_local_relation_extraction
        > __pycache__
        > logs
        { kg_1.json
        { kg_2.json
        { kg_3.json
        { prompt_1_examples.json
        ↓ prompt_1_template.md
        { prompt_2_examples.json
        ↓ prompt_2_template.md
        { run_1_triple_extraction.py
        { run_2_triple_refinement.py
        { run_3_triple_existence_check.py
      > s4_global_relation_extraction
      > s5_schema_generation
      > s6_kg_postprocessing
      > tests
      > utilities
      { clean_output.py
      { main.py
```

Figure 5.9: Python Project Structure

5 Overview of the Implementation of the Local Holistic KGC Pipeline

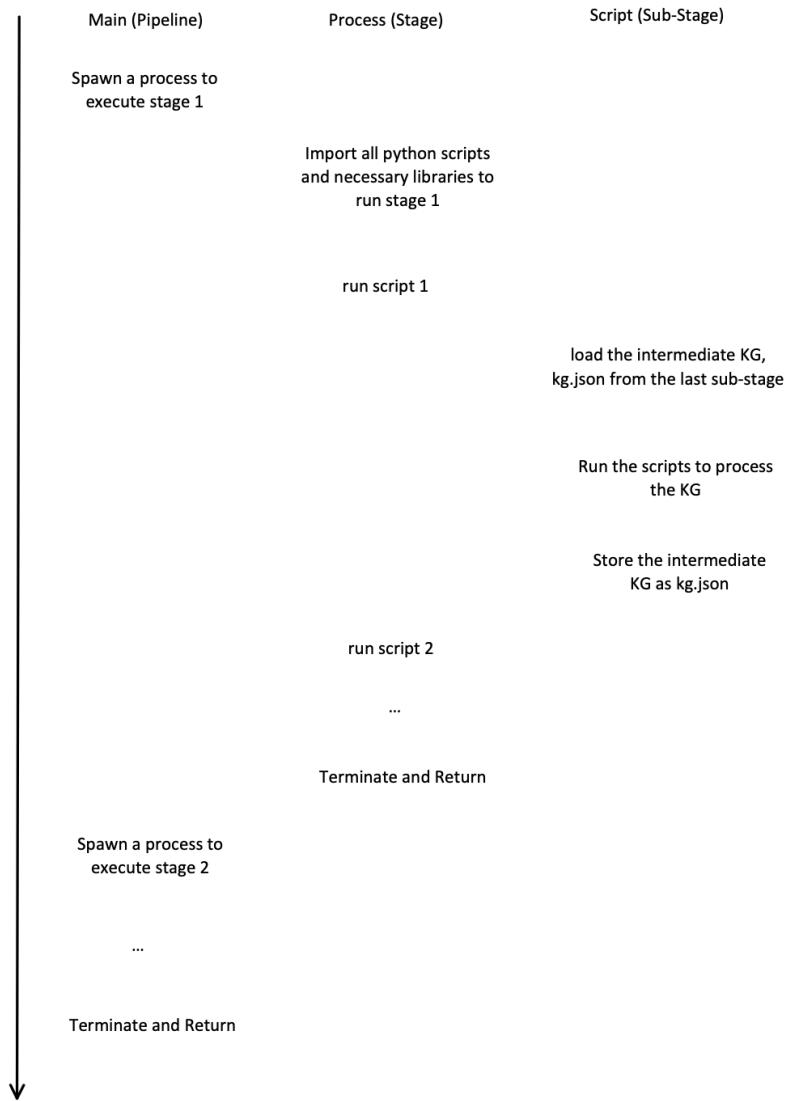


Figure 5.10: Execution Diagram of the Pipeline

5.4 Decision Made and Analysis

This section will summarise and analyse all decisions made about the structure of the pipeline.

5.4.1 Single Paper as the Primary Development Set

Pro:

- Accelerate the development process.
- Cover relatively diverse linguistic patterns compared to using only several sentences or paragraphs.
- It is a full paper. Some labelled KGC datasets do not even use a full paper, like SciERC, but only an abstract ([Luan et al., 2018](#)).

Con:

- Potentially make our pipeline biased to a single paper. Unlike traditional machine learning, no training is involved. However, the action of writing prompts by humans can be considered training. Since this paper has been read by us, our prompt can potentially be biased to this specific paper when we try to adjust our prompt to improve the performance.
- It may not be enough to discover more edge cases (i.e., the linguistic pattern that makes our pipeline fail to work as desired, e.g., cannot extract the named entity we want).
- It is unlabelled. i.e., no ground truth output KG to reference. During our research, we could not find such a labelled dataset with a mapping between a full academic paper and KG.
- Even though there are labelled datasets like SciREC and DocRED, they only consist of a limited number of predefined predicates and entity types. This does not align with our idea of open information extraction, as mentioned previously in Section [5.1.2](#).

5.4.2 Modular Design

We have kept our pipeline highly modular and stored each intermediate KG on disk instead of in memory for the following reasons:

- **Efficiency:** Different stages usually require different packages. During the execution of a stage, only the packages needed are loaded.
- **Upgradability:** Some stages can potentially be switched off (like the global relation extraction stage) and upgraded without affecting other stages.

5 Overview of the Implementation of the Local Holistic KGC Pipeline

- **Fault Tolerance:** Allow the pipelines to recover quickly in case of failure, e.g., a power outage.

The drawbacks include more disk space used, such as the code for loading and storing the intermediate files and the intermediate files themselves.

5.4.3 Using JSON instead of TTL

We use JSON instead of TTL as the format of the intermediate result of our pipeline for the following reasons:

- JSON is natively supported by Python and can be smoothly converted into Python Dictionaries in memory during the execution of the pipeline.
- Accessing the elements, e.g., an entity or sentence in Python Dictionary, only needs a simple indexing statement.
- Python Dictionary/JSON is ordered and hierarchical, which can better express the structure of a paper.
- Loading a TTL file requires the “rdflib” library to load it as an RDF-Graph Python object in memory. SPARQL queries are usually required to access elements, which can be more complex than simple indexing. SPARQL is a graph query language. It can be compared to SQL in relational databases.
- Nodes in RDF graphs have no sense of ordering. Therefore, every time we want to retrieve a list of sentences in order, we use SPARQL to sort it using SPARQL based on the index node, leading to higher complexity.
- JSON is likely more human-readable than TTL, which makes it easy for us to check the intermediate file directly without any graph visualisation tool.

The drawbacks include the extra step of converting between JSON and TTL in the first and the last stage, and our customised JSON-based format may not be easily understood by others who are already familiar with TTL.

The following compares the code for accessing Sentence 2 of Paragraph 2 of Section 2 using SPARQL (when loading TTL as RDF graph object directly) and indexing (when converting to the customised JSON-based format and loading as Python Dictionary), which demonstrates the complexity of using SPARQL for paper accessing in our scenario.

SPARQL

```
PREFIX askg-data: <https://www.anu.edu.au/data/scholarly/>
PREFIX askg-onto: <https://www.anu.edu.au/onto/scholarly#>

SELECT ?sentenceContent
WHERE {
  ?section a askg-onto:Section ;
            askg-onto:hasIndex 2 ;
            askg-onto:hasParagraph ?paragraph .
```

5.4 Decision Made and Analysis

```
?paragraph a askg-onto:Paragraph ;
    askg-onto:hasIndex 2 ;
    askg-onto:hasSentence ?sentence .

?sentence a askg-onto:Sentence ;
    askg-onto:hasIndex 2 ;
    askg-onto:hasText ?sentenceContent .
}
```

Indexing

```
paper["sections"] [2] ["paragraphs"] [2] ["sentences"] [2] ["text"]
```


Chapter 6

Stage I: Entity Mention Extraction, Classification, and Typing

The chapter will cover the implementation details of Stage 1 of the pipeline, Entity Mention Recognition, Classification, and Typing. Specifically, it will cover the algorithm, complexity and the key decisions made during the development.

6.1 Algorithm

Intuitively, as shown in Figure 6.1, Stage 1 of the pipeline operates by looping the paper document sentence by sentence, paragraph by paragraph, and section by section. In each iteration, a single sentence, paragraph, or section is sent to an LLM, and the entity mentions, along with its potential parent types, are extracted. This process is repeated three times with slightly different prompts so that the LLM can focus on “Named Entity Only”, “Named Entity plus General Concept”, and “All Potential Entity”. This not only ensures that there are minimal entity misses by focusing on different perspectives about entities but also allows the pipeline to know the specific class of entities, i.e., named entity, general concept, or other, through set subtraction. Having such entity classification may not be quite intuitive at this stage. However, this is due to some assumptions we have made in stage 3: Coreference Resolution and Entity Disambiguation (Section 7). Without having such a clear classification, our assumptions will fail. In the end, the pipeline checks whether each entity mention indeed appears in the original document, rather than imagining by the LLM. The input and output of this stage are shown in Figure 6.2, with the input components in grey and the output components in colour.

The detailed algorithm is as follows:

6 Stage I: Entity Mention Extraction, Classification, and Typing

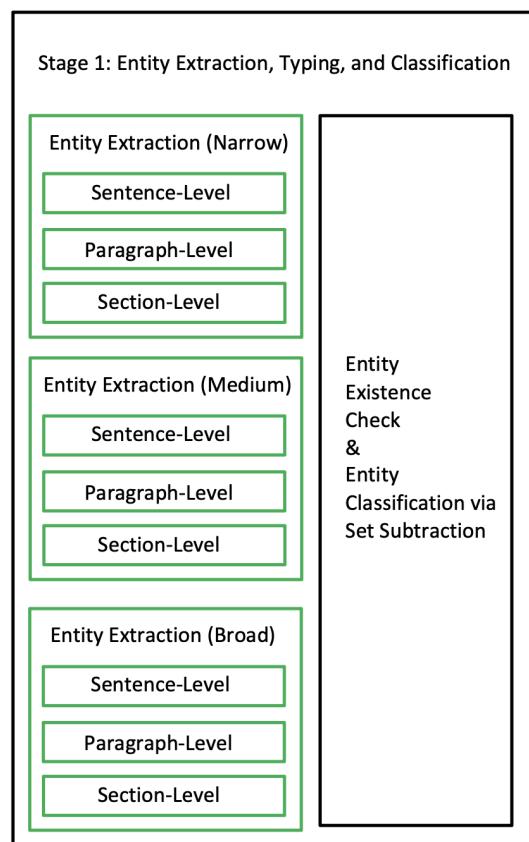


Figure 6.1: Entity Mention Extraction, Typing, and Classification (High-Level)

6.1 Algorithm

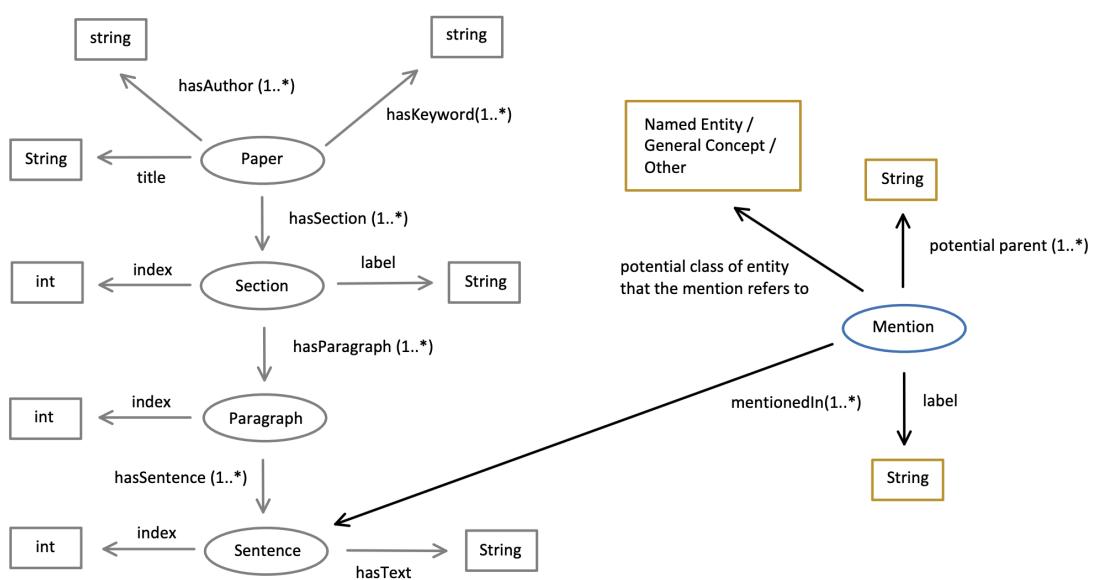


Figure 6.2: The input and output of Stage 1

6 Stage I: Entity Mention Extraction, Classification, and Typing

Algorithm

```
Input: A semi-structured paper
Output: An intermediate KG with the addition of Mentions nodes

# Extraction and Typing

For extraction_option in [Named Entity, Named Entity + General Concept,
Entity]

    For each section in the paper,

        For each paragraph in the section,

            For each sentence in the paragraph,

                * For the case of extracting Named Entities, before
                  prompting the LLM, inject a dummy Named Entity into the
                  sentence and remove it after the extraction to prevent the
                  LLM from hallucinating any Named Entities when a sentence
                  has no Named Entities

                [LLaMA Prompt_1] ask the LLM to extract all mentions
                referring to {extraction option} in the {sentence} and
                their potential parent types.

                Store them in the set called {extraction_option} directly
                under the specific {sentence} extracted from.

                [LLaMA Prompt_1] Similarly, but the {extraction_option} set is
                directly under the paragraph node.

                [LLaMA Prompt_1] Similarly, but the {extraction_option} set is
                directly under the section node.

# Clasificaton

For each section in the paper,

    For each paragraph in the section,

        For each sentence in the paragraph,

            its new "Named Entity" set = the original "Named Entity" set

            Its new "General Concept" set = the original "Named Entity +
```

6.1 Algorithm

```
General Concept" set - the new "Named Entity"

Its new "Other" set = the original "Entity" set - the new
"General Concept" set

Similarly.

Similarly.

# Existence Check and Reference Fixing

DEFAULT_NORMALISATION = lemmatisation using NTLK

def is_existing (mention, sentence):
    return whether the normalised mention is in the normalised sentence.

For each section in the paper,
    For each paragraph in the section,
        For each sentence in the paragraph,
            For each mention in {Named Entity} + {General Concept} +
{Other}:
                if not is_existing(mention, sentence),
                    discard it.

For each mention in {Named Entity} + {General Concept} + {Other}:
    For each sentence in the paragraph,
        if is_existing(mention, sentence),
            copy and add the mention to the specific set of the
            specific sentence.

Delete all mentions under the paragraph.

For each mention in {Named Entity} + {General Concept} + {Other}:
    For each paragraph in the section:
        For each sentence in the paragraph,
```

6 Stage I: Entity Mention Extraction, Classification, and Typing

```
if is_existing(mention, sentence),  
    copy and add the mention to the specific set of the  
    specific sentence.  
  
Delete all mentions under the section.  
  
# Moving all mentions from the sentence level to the paper level  
  
For each section in the paper,  
    For each paragraph in the section,  
        For each sentence in the paragraph,  
            For each mention in {Named Entity} + {General Concept} +  
            {Other}:  
                Move the mention to the paper level, i.e.,  
                paper["Mentions"]  
  
                While moving, maintain a link between the mention and the  
                original sentence from which it is extracted.  
  
                Also, maintain the information about the potential class of  
                entity that the mention refers to.
```

A key decision we made during the design of our algorithm was to inject a dummy Named Entity into the sentence during the Named Entity Mention Extraction. This was because, during development, we found that the LLaMA model we used suffered seriously from hallucination issues. When a sentence contains no Named Entities but only general concepts or other classes of entities, which is quite common (e.g., 'Cats eat fish'), the LLM then outputs anything else not belonging to Named Entities or even not in the original sentence. In other words, the LLM is “reluctant” to output an empty result. The injection is performed by appending a sentence like “This sentence is presented by “{an extremely rare person’s name}” at the end of the original sentence.

6.2 Prompts

The way we ask the LLM decoder extract mentions “Named Entity”, “Named Entity + General Concept”, and “Entity” is not about directly asking them along with these three keywords.

6.2 Prompts

Although in KGs, entities can almost refer to everything with an identity, narrowly speaking, the term “Entity” commonly refers only to “Named Entity + General Concept” in real life. Therefore, if we ask an LLM, especially the LLaMA model we used, to extract “Entities”, it will only extract “Named Entities + General Concepts”. This rule was found in multiple trial-and-error during the development. In addition, if we ask the LLM decoder to extract all “Mentions” in a prompt, it will then extract all “Entity” in our sense. Therefore, the mapping between the keywords during the extraction and the actual classes of entities that the extracted mentions refer to are as follows:

- Extracting Mentions to Named Entities -> Extracting “Named Entities”
- Extracting Mentions to Named Entities and General Concepts -> Extracting “Entities”
- Extracting Mentions to All Entities -> Extracting “Mentions”

The actual prompt template the pipeline used is as follows:

Prompt 1: Entity Extraction

Task Definition

```
You are a linguistic expert involved in a {named entity recognition/entity extraction/mention extraction} task for academic papers. Given a text in a paper, your task is to identify and extract all {named entities/entities/mentions} and assign them one or more types.
```

Output Format

```
Please write your answer in JSON format, consisting of zero or more entity-type pairs, as shown below:
```

```
"""
{format_example}
"""
```

Examples

```
**Input**: {input_example_1}
```

```
**Output**:
```

```
"""
{output_example_1}
"""
```

```
**Input**: {input_example_2}
```

```
**Output**:
```

```
"""
{output_example_2}
"""
```

6 Stage I: Entity Mention Extraction, Classification, and Typing

```
**Input**: {input_example_3}
**Output**:
```
{output_example_3}
```

## Your Task

**Input**: {text}
**Output**:

*Please write your answer*
```

Along with the instructions, we have also prepared three hand-crafted examples in order to show the LLM clearly what it is supposed to do. These examples are fixed for different sentences, paragraphs, and sections but are specific to different extraction substages, e.g., “Named Entities”, “Named Entities + General Concepts”, and “All Entities”. We set out the background at the beginning to ensure the LLM responds as a linguistic expert.

6.3 Assumptions Made

Several key assumptions were made during the development of this stage. Some of them also apply to later stages.

6.3.1 A1: LLMs are Knowledgeable

We assume that the LLM used knows all the common terminologies in NLP and linguistics. Therefore, our prompt can eliminate their explicit definitions, e.g., what a Named Entity is, in order to make it concise and reduce the complexity of an LLM call. *This assumption applies to all stages in the pipeline.*

6.3.2 A2: All Sections Fit the Context Limit of the LLM Used

We assume that the input paper has been well-preprocessed so that all its sections can fit the context limit of the LLM used by the pipeline. In addition, we also assume that every section plus the abstract of the paper (the first section) also fit within the context limit of the LLM. Having sections of appropriate length is common in academic writing, especially for published papers. If it is calculated by the pipeline that a section exists that exceeds the context limit of the LLM, the pipeline will abort, leaving its intermediate KG for recovery. *This assumption applies to all stages in the pipeline.*

6.3.3 A3: Identical Mentions in a Local Text Refer to the Same Thing

In our prompt, we ask the LLM to extract mentions in a given local text (a sentence, paragraph, or section) and then output a list of mentions found. However, we do not

6.3 Assumptions Made

track the specific location of the mentions. Therefore, we assume that all mentions with the same name within a single sentence refer to the same thing (entity). For example, consider the sentence, “An apple is the fruit of the apple tree.” two mentions of “apple” refer to the same concept. However, we acknowledge that in some rare cases, this assumption can break. For example, consider the sentence, “Apple is a technological company whose LOGO is a Bitten Apple”, if it turns out that “Apple” is extracted as a mention. there will be no way for our pipeline to figure out the exact meaning of “Apple”. Nevertheless, academic writings are unlikely to consist of ambiguous terms. If it ends up with two mentions referring to different things in the same sentence, e.g., “(Wang et al., 2024a)” and “(Wang et al., 2024b)”, we expect that the authors have disambiguated them.

The potential way to mitigate this assumption is to ask the LLM to also output the position in terms of the token index or character index of each mention, or instead of generating a list of mentions, ask the LLM to annotate the original with some symbol. The former can be hard to implement because the output of LLM is somewhat stochastic. LLM decoders operate by predating the next tokens continuing the previous tokens. Therefore, they may struggle to perform tasks requiring high accuracy (Pan et al., 2023), like counting and doing maths. Annotating the entity mentions also has disadvantages. This is because if entity mentions are nested or overlapped with each other, then annotation can become complex. Therefore, we made this assumption in our pipeline to keep things simple.

For two identical mentions in different sentences, since our data structure and the pipeline can track their locations and know their differences in locations, they do not rely on this assumption and can refer to different things.

6.3.4 A4: An Entity Mention Extracted at the Paragraph or Section Level Means All Sentence Containing the Term Same As the Entity Mention Have the Entity Mention

This assumption may not be quite intuitive. However, it basically means that if an entity mention, e.g., “ANU”, is extracted from the paragraph/section level, as long as the sentence within the paragraph/section contains the term “ANU”, the sentence will also obtain such mention, no matter mention “ANU” has been extracted from the sentence level or not.

Another example is that if the entity mention “It” is extracted from a paragraph/section, and if there are multiple “It” within the paragraph or section, we will put the mention “It” into all the sentences containing “It”. This is because having the entity “It” at the paragraph or section level is too ambiguous. We have no idea if “It” comes from sentence 1 or sentence 10. Therefore, we simply put “It” in all sentences containing “It” within the paragraph/section. By doing so, we effectively split a paragraph or section-level entity mentions into multiple sentence-level mentions. Before coreference resolution, we treat them differently and assume that they refer to different things. This is because

6 Stage I: Entity Mention Extraction, Classification, and Typing

even within a paragraph, not all identical terms, such as “It” necessarily refer to the same thing.

6.3.5 A5: All Named Entities and Concepts Extracted Are Indeed Named Entities and Concepts

We assume that when we ask the LLM to extract Named Entities, it will only return Named Entities. When we ask the LLM to extract Named Entities plus General Concepts, it will only return Named Entities plus General Concepts. This is important because our classification via the set subtraction approach relies on the correctness of the extraction process.

6.4 Limitations

There are two major limitations in this stage.

6.4.1 L1: Only Mentions That Syntactically Appear in Text Are Extracted

During the development, we found that the output of LLMs can sometimes be stochastic even with a low-temperature parameter. For example, if the original sentence is, “There are 5 people.” The output of the LLM can be “Five” instead of “5”. However, if the normalisation tool is not advanced enough, i.e., turning both “Five” and “5” into some normalised form, the mention “Five” will be rejected by our pipeline, leading to some entity miss. However, we still check if a mention extracted syntactically appears in its original text because the LLM can indeed extract something that is hallucinated by itself. Therefore, this is a trade-off.

[Wang et al. \(2023\)](#)’s study for avoiding hallucinating is that, for each entity mention extracted, prompt the LLM again to check if it is indeed a mention in the original sentence. However, not only can this lead to high complexity, but the checking result itself can also be hallucinated by the LLM, i.e., we cannot be sure that answer of “Yes” in an LLM response indeed means the entity mention exists in the original sentence.

Therefore, we tend to use the most traditional rule-based method to avoid hallucination, which leads to this limitation.

6.4.2 L2: Only Entities That Have at least One Mentions Are Captured

Our general KGC approach is to extract mentions and, from these mentions, construct entities. However, not all entities have a syntactical form in the original sentence. Like the entity “moment” in the sentence “Amy was practising piano at 5 PM”. This limitation already exists during the KGC pipeline design, as discussed in the pipeline design chapter ([4.4.2](#)), and carries through the implementation.

6.5 Complexity Analysis

This section will outline the general complexity analysis approach for all stages of the pipeline and derive the complexity of Stage 1.

6.5.1 Analysis Approach

Since an LLM decoder call can be dominantly expensive. During the modelling of our pipeline complexity, our complexity will only based on the number of times an LLM decoder is called, while other complexities, such as regular computations, file IO, or even LLM encoder calls are all modelled as $O(n)$.

Therefore, Stage 0: Input Preprocessing and Stage 6: KG Post-Processing both have a complexity of $O(n)$.

Although a single LLM decoder call is more than quadratic with respective to the number of tokens in a prompt as mentioned in the background section (2.4). Since we are loop through the document sentence by sentence, paragraph by paragraph, or section by section. The lengths of prompts are bounded and usually have similar lengths. Therefore, in our analysis, we will simply treat every LLM decoder call as $O(1)$ this will apply all the complexity analysis in the later implementation chapters.

All complexity modelled in the implementation chapter (Chapter 6 to 10) will be confirmed with the actual runtime of paper with different lengths.

6.5.2 Stage 1 Complexity

Since the algorithm of this stage loops through the paper sentence by sentence, paragraph by paragraph, and section by section, the number of times an LLM decoder is called is approximately proportional to the length of the paper L . This algorithm scans through the whole document three times in order to extract entities at three hierarchical levels. Therefore, more precisely, the complexity is $O(9L)$.

6.6 Decisions Made and Analysis

The following subsections are about the key decision made and their reason.

6.6.1 Simple Prompts vs. Detailed Prompts

We tend to write prompts as simply and concisely as possible. This applies to all stages of the pipeline. This is because LLM complexity is more than quadratically proportional to number of tokens in the prompt. During our development, we indeed created both a short version and a long version for the prompts. However, it turned out that an LLM could respond to a short prompt within a second. In contrast, a long prompt can take up to half a minute, which significantly affects the runtime performance since our pipeline makes plenty of prompts to the LLM during its execution.

6 Stage I: Entity Mention Extraction, Classification, and Typing

In addition, different people may have slightly different understandings of some terminologies, such as named entities. This also applies to humans and AIs. If we prepare a long prompt trying to define what a named entity is, but such a definition contradicts the internal knowledge of the LLM, the LLM may get confused. Therefore, we leave the definition of terminologies to the LLM and trust the LLM. However, the drawback is that we have to assume LLMs are knowledgeable.

The decision above applies to all the later stages.

Although [Xie et al. \(2023\)](#)' study suggests that we can add part-of-speech tagging along with the original text to assist the LLM in extracting the named entity based on both semantic and syntactical reasoning. However, during our development, we have found that adding Part-of-Speed only leads to minimal impacts. However, since the prompt became longer, the runtime significantly increased. Therefore, we decided to keep our prompt simple and expect the LLM to do the task without the assistance of Part-of-Speech tagging.

[Wang et al. \(2023\)](#)'s study suggests that the supplied examples in the prompt should be relevant to the input text. For example, if we want to extract the named entities from the sentence, "ANU is a university located in Canberra." the supplied examples in the prompt should be drawn from some knowledge base that is semantically relevant to the input sentence. However, most currently existing datasets that can serve as a knowledge base for entity extraction only consider entities that fall within a predefined set of types. Therefore, these examples can be misleading because our goal is to let LLM extract anything from the sentence as long as it is an entity without restricting it to some predefined types. Therefore, the use of a knowledge base to retrieve relevant examples is not practical during open entity extraction. Instead, we decided to provide only three general and fixed hand-created examples. The main goal is to only let the LLM be familiar with the task definition instead of teaching precisely what is supposed to be extracted as an entity and what is not. This is also another decision regarding making prompts as simple as possible.

6.6.2 Hierarchical Extraction

Our pipeline performs entity extraction hierarchically in two dimensions: entity and text.

Extracting entities through all sentence, paragraph, and section levels allows the LLM to have both a narrow view and a broader view of the text so that there are fewer entities missed.

Figures [6.3](#), [6.4](#), and [6.5](#) show the comparison between sentence-level extract extraction and paragraph-level entity extraction for the same part of the text of the paper in our development set. Figures [6.3](#) and [6.4](#) consist of the sentences from the paragraph contained in Figure [6.5](#). We would simply refer to them as sentence 1, sentence 2, and paragraph 1, respectively.

6.6 Decisions Made and Analysis

Your Task

Input: This manuscript introduces J2RM: a tool to process mappings from JSON data to RDF triples guided by an OWL2 ontology structure.

Output:

Please write your answer



GPT4ALL Llama 3 8B Instruct

Here is the output in JSON format:

```
[  
  {  
    "entity": "J2RM",  
    "types": [  
      "tool",  
      "software"  
    ]  
  },  
  {  
    "entity": "JSON data",  
    "types": [  
      "data",  
      "format"  
    ]  
  },  
  {  
    "entity": "RDF triples",  
    "types": [  
      "data",  
      "format"  
    ]  
  },  
  {  
    "entity": "OWL2 ontology structure",  
    "types": [  
      "ontology",  
      "knowledge representation"  
    ]  
  }  
]
```

Let me know if you have any questions or need further clarification!

Figure 6.3: Sentence-Level Entity Extraction

Note that the prompt is only partially captured in the screenshot due to the size, but they are similar to the prompt template shown previously. The tool we used is the GUI version of GPT4ALL. Although our pipeline uses the Python package version of GPT4ALL, They function similarly.

For sentence 1 shown in Figure 6.3, it was found that most entities are extracted except “manuscript”. However, “manuscript” was captured by the paragraph-level extraction in Figure 6.5.

In addition, many entities are missed during the paragraph-level extraction, such as “RDF-Graph creation”. However, the mention “RDF-Graph creation” was successfully captured by the sentence-level extraction as shown in Figure 6.4.

Therefore, the different hierarchical levels of entity extractions can complement each other, similar to the hierarchical entity extraction from narrow entity class to board entity class. However, the key reason for the hierarchical entity extraction of different broadness is to perform entity classification via set subtraction.

6 Stage I: Entity Mention Extraction, Classification, and Typing

Your Task

Input: They are embedded in an ontology file so that they can be readily deployed and shared to automate RDF-graph creation. **Output:**

Please write your answer



GPT4All Llama 3 8B Instruct

Here is the output in JSON format:

```
[  
  {  
    "entity": "ontology file",  
    "types": [  
      "file",  
      "data source"  
    ]  
  },  
  {  
    "entity": "RDF-graph creation",  
    "types": [  
      "process",  
      "algorithm"  
    ]  
  }  
]
```

Figure 6.4: Sentence-Level Entity Extraction 2

6.6.3 Explicit Entity Classification vs. Classification via Set Subtraction

During the development, we found that directly asking the LLM to classify entities into “Named Entity”, “General Concept”, and “Other” does not work well. Unlike LLM encoders, LLM decoders struggle to perform classification tasks. This is mentioned in [Edwards and Camacho-Collados \(2024\)](#) and [Sun et al. \(2023\)](#)’s study. This is why we choose to perform the implicit classification via set subtraction.

We have tried two explicit entity classifications. One is the classified entities when they are extracted. That is, in the prompt, we explicitly ask the LLM to extract either Named Entities, General Concepts and Other that do not belong to Named Entity and General Concept. However, this approach does not work because we found that the more strict and specific the instruction, the worse the performance the LLM has, which is likely due to the hallucination issue. As mentioned earlier. Extracting named entities only is already difficult for the LLM, as we have to inject a dummy named entity to ensure the LLM does not hallucinate. Therefore, extracting general concepts or other entities only can also be hard.

The second way to perform explicit entity classification is to first prompt the LLM to extract all entities regardless of their class. Then, we make another prompt to ask the LLM what the entity belongs to, either named entities, general concepts, or others. It ended up that the LLM could easily make wrong predictions.

Therefore, our final solution is to gradually increase the scope of the entity required for the LLM to extract. In this way, not only do we perform a basic entity classification via

Your Task

Input: This manuscript introduces J2RM: a tool to process mappings from JSON data to RDF triples guided by an OWL2 ontology structure. The mappings are defined as annotation properties associated with each ontology entity of interest. They are embedded in an ontology file so that they can be readily deployed and shared to automate RDF-graph creation. In this paper, we motivate the need for such mappings, describe some of their definitions on a use case example, present the formal grammar of the mapping language, and explain how these mappings work. We conclude with a discussion of the key aspects, main contributions, and future improvements. **Output:**

Please write your answer



GPT4All Llama 3 8B Instruct

Here is the output in JSON format:

```
[
  {
    "entity": "J2RM",
    "types": [
      "tool",
      "software"
    ]
  },
  {
    "entity": "JSON data",
    "types": [
      "data",
      "format"
    ]
  },
  {
    "entity": "RDF triples",
    "types": [
      "data structure",
      "semantic web"
    ]
  },
  {
    "entity": "OWL2 ontology structure",
    "types": [
      "ontology",
      "knowledge representation"
    ]
  },
  {
    "entity": "annotation properties",
    "types": [
      "metadata",
      "data annotation"
    ]
  },
  {
    "entity": "ontology entity of interest",
    "types": [
      "concept",
      "knowledge concept"
    ]
  },
  {
    "entity": "manuscript",
    "types": [
      "academic paper",
      "research document"
    ]
  }
]
```

Figure 6.5: Paragraph-Level Entity Extraction

6 Stage I: Entity Mention Extraction, Classification, and Typing

set subtraction, but we also mitigate the shortage of classification in LLM decoders.

6.6.4 JSON Output vs. Customised-Format Output

Last but not least, because the response of LLM decoders is plain text, we have to parse the response back to structured data type, such as a list of entity-type pairs, in order for the algorithmic part of the pipeline to process.

Originally, we defined our customised format for the output in the prompt, i.e., using “[” and “]” or some special to mark a list so that our parser can successfully parse the entity lists in the LLM’s response. However, it concluded that there always exists some edge case that prevents successful parsing. Later, we found that LLM decoders seem to be good at providing JSON output, and the parsing tool from JSON into Python Dictionary has already been built into it. Many edge cases can also be automatically handled by JSON through escaping. Therefore, we decided to make the response of all LLM calls in JSON format.

In addition, we decided to use the Markdown syntax to write all the prompts in the pipeline because we found that the LLM decoder used, i.e., LLaMA, can properly understand and handle Markdown. There are also studies showing that a more structured syntax can help an LLM understand the prompt better compared to plain text ([Wang et al., 2024](#)).

Chapter 7

Stage II: Entity Disambiguation and Coreference Resolution

This chapter will cover the implementation details of Stage 2: Entity Disambiguation and Coreference Resolution by discussing the algorithm, complexity, and key decisions made.

7.1 Algorithm

Intuitively, as shown in Figure 7.1, the algorithm starts by creating an entity node in the intermediate KG for all mentions. That is, it conservatively treats all mentions as different things, e.g., “Python” in sentence 1 is different from “Python” in sentence 14. Therefore, every entity only has one mention, for now, where the entity’s label (name) is the same as the mention’s label.

Then, Entity Disambiguation is performed. This step is about carefully merging two entities with the same name. It is mainly based on the assumption that “Named entities

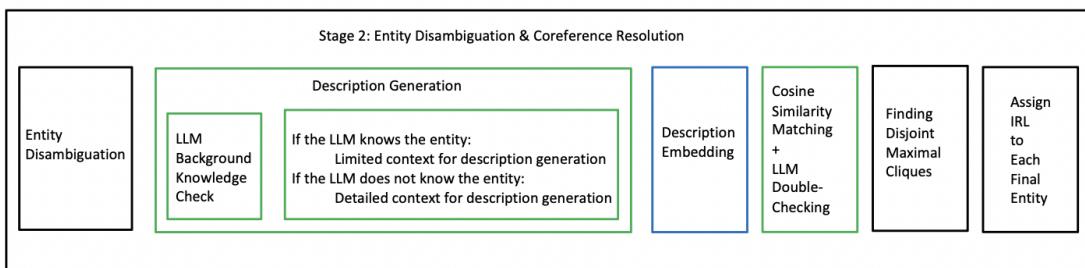


Figure 7.1: Entity Disambiguation and Coreference Resolution (High-Level)

7 Stage II: Entity Disambiguation and Coreference Resolution

and general concepts in academic writing are unambiguous”. For example, we assume that all occurrences of “Python” in academic writing refer to either a type of snake or a programming language. All occurrences of “Wang” refer to the same person. The general concept “Plane” refers to either a flat surface or a flying vehicle, but not both. We expect that the author of the paper has appropriately handled any ambiguity, e.g., adding brackets for any potentially ambiguous term. Therefore, with this assumption, we can safely merge all entities belonging to named entities and general concepts with the same name. When merging two entities, their mentions set are combined. The entity label will be the label of the mention that appears first in the paper. This is also based on the assumption that academic writing tends to only provide the full name of a term, especially a terminology, at the beginning and use its acronym throughout the rest of the text. However, the full name should be the most representative name of an entity. Other names that are not used as labels then become aliases.

Even though we merge named entities and general concepts, we keep all Other entities untouched because our assumption does not hold for all types of entities. For example, consider an entity “It” created from a mention in the original paper. Without Coreference Resolution, we cannot be sure if it is a named entity, general concept or neither. An “It” in sentence 1 is clearly not necessarily the same as the entity “It” in sentence 14. If a named entity is merged with a concept, we treat it as a named entity.

After merging entities with the same name, we then try to merge entities with different names but refer to the same thing, i.e., coreference resolution. Like what LLM-based coreference resolution is commonly performed, as discussed in the literature review section (3) our coreference resolution is based on comparing the embeddings of any two entities. If they are highly similar and the result is double-checked by the LLM decoder, we then draw a temporary link between them. However, this does not mean they must be merged into the same entity. This is because if A is similar to B and B is similar to C, it does not mean A is similar to C since the nuances between two entities cannot be easily detectable, and there exists a threshold between treating them as two different things or the same thing. Our algorithm groups multiple entities into the same entities only if their temporary links form a complete subgraph, and we also hope the groups are as large as possible, i.e. finding maximal cliques. However, we also hope each clique is disjointed. Therefore, we iteratively call the Bron-Kerbosch algorithm ([Bron and Kerbosch, 1973](#)) to find the maximum cliques and remove all the nodes belonging to the clique in the main graph. After merging, we assign a unique IRI for each entity and mention it so that our intermediate KG can eventually become a valid RDF-KG.

To obtain embedding for each entity, we use the entity labels, aliases, potential parent types, and most importantly, its description. A good description should clearly define what an entity is and be able to distinguish it from other entities with the same name. It should also provide essential background knowledge about the entity if the entity is newly introduced by the paper and unseen by the LLM.

To generate a high-quality entity, we first perform a background check for the LLM

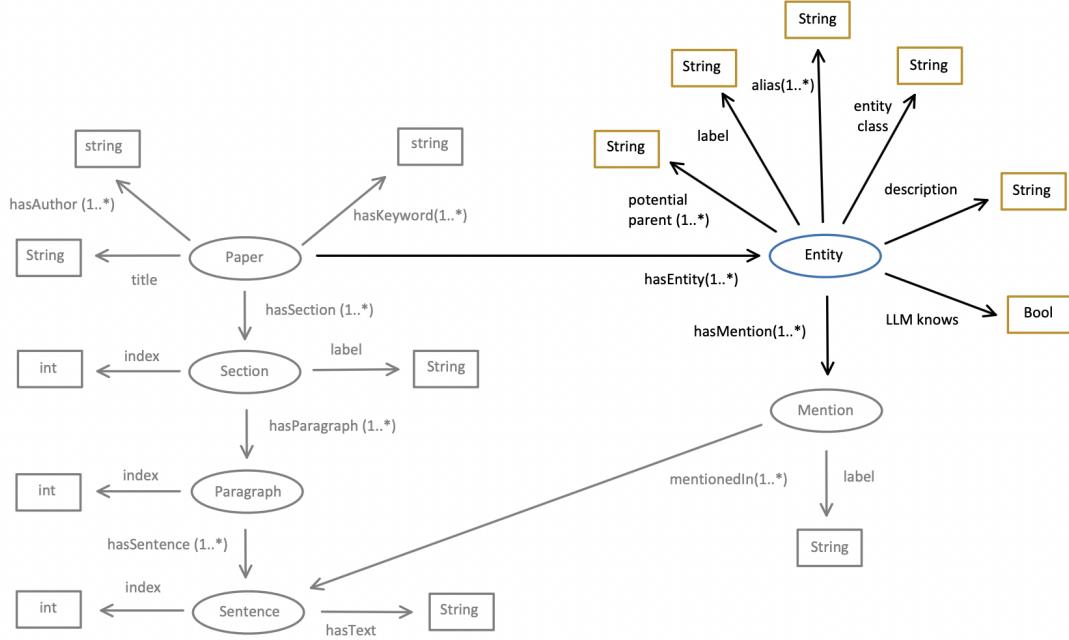


Figure 7.2: Input and output of Stage 2

to see, for each entity, if the LLM knows it without the assistance of a context. This step can effectively mark all new entities, e.g., any new terminologies introduced by the paper and unseen by the LLM, as well as any potentially ambiguous entity, such as “It”. When generating a description, if the LLM knows the entity, limited context will be given to the LLM in order for the description to be as general as possible, such as the entity “University”. If the LLM does not know the entity, detailed context will be given, including the title, authors, and abstract of the paper, as well as the specific section where the entity is located, such as the entity “J2RM” introduced in the paper in our development set. For Other entities, no matter if the LLM is familiar with it, our pipeline always supplies it with detailed context. In this case, with sufficient recourse, the description for any new or ambiguous entity, like “J2RM” can achieve a high quality. When we embed an entity along with its description, the embedding is then also representative.

The input and output of this stage are shown in Figure 7.2, with the input components in grey and the output components in colour.

7 Stage II: Entity Disambiguation and Coreference Resolution

The detailed algorithm is as follows:

Algorithm

```
Input: An intermediate KG with Mention nodes
Output: An intermediate KG with the addition of Entity nodes

# Entity Creation

For each mention in the KG,
    create an entity node, where its label is the same as the label of the
    mention, its alias set contains only its label, its potential parent
    set is the same as the mention's one, and its entity class is the same
    as the potential entity class of the mention.

    Create a link between the newly created entity node and the original
    mention node.

    Delete all attributes of the original mention node except its label,
    serving as the local name of the entity in a specific sentence.

    Create a description attribute for the entity as left empty.

# Entity Merging Function

def merge_entity(entity_1, entity_2):
    new_potential_parents = union of the entity_1's parents and entity_2's
    new_mentions = union of the entity_1's mentions and entity_2's
    new_aliases = union of the entity_1's aliases and entity_2's

    new_label = the label of the Mention in new_mentions that is mentioned
    first in the paper.

    new_description = the description of the Entity with the Mention in its
    original_mention set that is mentioned first in the paper.

    new_entity_class = the original entity classes of the two entities with
    the high precedence. (named entity > general concept > other)

    return construct a new entity with the new attributes and delete the
    two old entities.

# Entity Disambiguation

For entity_1 in the KG,
    for entity_2 in the KG,
```

7.1 Algorithm

```
if entity_1 != other and entity_2 != other,  
    if entity_1["label"] == entity_2["label"],  
        draw a temporary link between them.  
  
Merge all entities connected by temporary links using merge_entity.  
  
# LLM Background Knowledge Check  
  
For each entity,  
    [LLaMA Prompt_1] Given the entity's label only, ask the LLM if it can  
    confidently know the exact entity the entity label refers to.  
  
# Description Generation  
  
For each entity in the KG,  
    if entity["LLM_knows"] == False or entity["class"] = "Other",  
        [LLaMA Prompt_2.1] Use the entity's label, potential parent types,  
        the paper's title, the paper's authors, the paper's keywords, the  
        paper abstract (first section), the section-level context (the  
        section where the entity appears), the sentence-level context (the  
        specific sentence in the section where the entity appears) to  
        generate a short description.  
    else,  
        [LLaMA Prompt_2.2] Use the entity's label, potential parent types,  
        sentence-level context to generate a short description.  
  
# Embedding Generation  
  
For each entity in the KG,  
    [BGE] use its label, potential parent types, and description to  
    generate an embedding.  
  
# Coreference Resolution  
  
For entity_1 in KG["Entities"],
```

7 Stage II: Entity Disambiguation and Coreference Resolution

```
for entity_2 in KG["Entities"] ,  
    if entity_1_id < entity_2_id,  
        compute the cosine similarity between the embeddings of  
        entity_1 and entity_2.  
  
    If similarity > 0.9,  
  
        [LLaMA Prompt_3] Given the paper's title, authors, and  
        keywords, given the labels, potential parent types,  
        description, section-level context, and sentence-level  
        context of both entities, ask the LLM to determine if they  
        are the same.  
  
        * If having two section-level contexts is too long for the  
        context limit, use paragraph-level contexts instead.  
  
    If the LLM says yes,  
  
        Create a temporary link between the two entities.  
  
# Entity Merging  
  
disjoint_maximal_cliques = []  
  
In the graph of related entities, use the Bron-Kerbosch algorithm to find  
a set of (non-disjoint) maximal cliques.  
  
Select the largest clique within the set of maximal cliques.  
  
Remove all the nodes in the original related entity graph that are in the  
selected maximal clique.  
  
Add the selected maximal clique into disjoint_maximal_cliques.  
  
Repeat until no more maximal cliques can be found with sizes larger than 1.  
  
For each clique in disjoint_maximal_cliques,  
    merge all entities using merge_entity.  
  
# Assign IRIs  
  
For each entity,
```

assign it an IRI based on its label for readability but ensure uniqueness.

assign an IRI for all its mentions based on the IRI of the entity.

7.2 Prompts

The following are the simplified versions of the three LLM prompts mentioned in the algorithm.

Prompt 1: LLM Knowledge Check

Task

Assume that you are reading an academic paper. Given a term extracted from the paper and its types, without seeing the context, can you tell me what the term "{term}" refers to based only on your current knowledge?

- Answer true if you are confidently sure what it is.
- Answer false if you do not know the term, or it is potentially ambiguous, i.e., it may have several meanings, but its specific meaning cannot be determined without seeing the context.

Output Format ...
Term ...
Answer ...

Prompt 2.1: Description Generation (Detailed)

Task

You are a linguistic expert involved in a description writing task for an entity mentioned in a paper. Given an entity's mention "{label}", its types, the abstract of the paper (paper-level context), the specific section where the mention appears (section-level context), and the specific sentence of the entity mention (sentence-level context), your task is to write a single-sentence description that can precisely tell what the actual entity the mention "{label}" refers to.

Output Format ...
Input ...
Output ...

Prompt 2.2: Description Generation (General)

Task

You are a linguistic expert who is involved in a description writing task for an entity mentioned in a paper. Given an entity's mention "{label}", its types, and the specific sentence of the entity mention (context), your task is to write a single-sentence description that can precisely tell what the actual entity the mention "{label}" refers to.

- Your description should be as general as possible and not be specific to the given context.
- The context is mainly used to help you understand the entity or disambiguate the entity with others with the same name.

Output Format ...

Input ...

Output ...

Prompt 3: LLM Double Check

Task

You are a linguistic expert who is involved in the task of disambiguating two terms in a paper. Given the abstract of the paper (paper-level context), the term "{label_1}" and the term "{label_2}", each with its label, types, description, and the specific section where the mention appears (section-level context), and the specific sentence of the entity mention (sentence-level context), your task is to determine whether these two terms refer to **exactly** the same entity or concept. Please provide a short explanation and then answer true or false in JSON format, e.g., ...

Input ...

Output ...

7.3 Assumptions Made

There are a number of assumptions made in this stage:

- When merging entities by names, we assume that all named entities and general concepts in the paper are unambiguous, i.e., named entities with the same label always refer to the same thing, which is highly likely in academic writing.
- When deciding the label of an entity, we assume that the mention of an entity that appears first in the paper always carries the most representative name for the entity.
- When using the LLM to generate a description for an entity that the LLM does

7.4 Limitations

not know, we assume that giving the paper’s title, authors, keywords and the entity’s label, types, section-level context, and sentence-level context is enough for the LLM to understand the entity so that the description generated is accurate.

- When merging two entities and deciding the description of the new entity, we assume that the description of the original entity with its mention that first appears in the paper is the most representative description.
- When merging two entities and there is a conflict between the entity class of the new entity, we assume that classifying as Named Entity is always more accurate than classifying as General Concept, which is more accurate than classifying as Other Entity.

7.4 Limitations

Several limitations can be found during this stage.

- We used embeddings to check if two entities are the same. However, the embeddings themselves can be lossy. That is, two close embeddings may not always represent two close entities in the real world.
- We used description to generate embeddings for entities. However, the descriptions generated by the LLM decoder are already lossy.
- When checking the LLM familiarity of an entity, the LLM may not answer honestly if it knows the entity or not. It can be too optimistic or conservative. This is because the behaviours of LLMs are stochastic, and they may suffer from the hallucination issue ([Pan et al., 2023](#)).
- When asking the LLM if two entities are the same during the double-checking process, the LLM may not answer honestly if two entities are the same. It can be too optimistic or conservative.

7.5 Complexity Analysis

The algorithm involves looping through all entities (not fully merged) twice to check if the LLM are familiar with them and generates a description. Therefore, the complexity here is $O(2M)$, where M is the number of mentions. This the process of entity merging has yet finished before this stage. We choose to use the number of mentions instead of the number of entity. Even though we can create an entity for each mention at the beginning of this stage, the number of entities is not yet final. That is why we use M instead of E .

Although the process of generating embedding for entities involves LLM encoders, since LLM encoders are generally fast and can be done in a second during the development,

7 Stage II: Entity Disambiguation and Coreference Resolution

we simply ignore their complexity. Similarity for the pairwise comparison for their embeddings.

However, although we only ask the LLM decoder to double-check only when two entities are found to be highly similar, in the worst-case scenario, it is still $O(E^2)$. Therefore, the overall complexity of this stage is $O(2E + E^2)$.

7.6 Decisions Made and Analysis

Two major decisions were made during the development of this stage. One is about the use of description in generating embedding. The other is about using disjoint maximal cliques instead of connected subgraphs to determine if a group of entities should be merged.

7.6.1 Using Description for Embedding Generation

During the implementation of the pipeline, we found that the description of an entity is the most valuable resource for Entity Disambiguation and Coreference Resolution. This is because even though two entities have the same name, like “Apple” (a company) and “Apple” (a fruit). Their description should be clearly different. The second most valuable resource is types. Clearly, the two “Apples” in the example have two different types. Our findings are further confirmed by some literature about entity disambiguation and coreference resolution, such as [Salman et al. \(2024\)](#), [Li et al. \(2020\)](#), and [Ayoola et al. \(2022\)](#)’ studies, where entity description or entity types are used when distinguishing two entities.

Traditionally, when Entity Disambiguation and Coreference Resolution are performed solely based on LLM encoders, the context may also be used directly to generate embedding ([Joshi et al., 2020](#)). However, this requires some fine-tuning for the LLM encoders because the general-purpose encoder we used, i.e., BGE and M3, can only embed the input as a whole. Therefore, if the context is used, the LLM may be distracted, i.e., focusing on generating the embedding for the context instead of the entity itself. Since our research tends to do everything purely based on in-context learning without fine-tuning any models, we choose to use description for embedding generating. In this case, everything input to the general-purpose LLM encoder is highly relevant to the entity. Therefore, the embedding generated should also accurately represent the entity.

The generated description is helpful not only in Coreference Resolution and Entity Disambiguation but also in the later stages, such as relation extraction. Essentially, whenever we supply the LLM with an entity, we always attach its description and sometimes its types. This is because only the entity label (name) is insufficient for the LLM or even humans to determine what it is. The description can also help the LLM to understand the LLM even though the entity is new to the LLM.

7.6.2 Disjoint Maximal Cliques vs. Disjoint Connected Subgraphs

In our initial implementation, we merge any two entities as long as they are related, i.e., based on connected subgraphs. However, the result is that many clearly distinct entities are merged together, such as “RDF-Graph” and “J2RM” (a tool to convert JSON into RDF). This is because there are often nuances between two entities with different names. However, if A and B are close enough to become the same entity and B and C are also close enough, it does not necessarily mean A and C are close. The worst case is that multiple entities form a hyper-line in the vector space of their embeddings, where the entities at either end are clearly different. Therefore, we merged a group of entities only when they formed a clique in the related entities graph. This is similar to clustering. However, we strictly control the diameter of each cluster.

Chapter 8

Stage III: Local Relation Extraction

This chapter will present the implementation details of Stage 3: Local Relation Extraction in a similar way to the previous chapters, i.e., the algorithms, prompts, assumptions made, limitations, complexity, and decisions made.

8.1 Algorithm

The way how local relation extraction works at a high level is similar to entity extraction. As shown in Figure 8.1, the algorithm starts by looping through the paper sentence by sentence, paragraph by paragraph, and section by section. Within each iteration, a text (sentence/paragraph/section) along with all the entities that have mentions linking to the text is input to the LLM. The output of the LLM is a list of triples, where the subject and the object come from the given entities and the predicate is extracted on the fly. During the implementation, we found that the LLM model we used, i.e., LLaMA, can sometimes return only the subject and the predicate, where the predicate consists of both the verb and the object, leaving the actual object slot of the triple empty. This is potentially because the term “predicate” is ambiguous and may refer to the “verb” only or the combination of “verb” and “object”. However, even if we explicitly ask the LLM to extract (subject, verb, object) triples, the problem persists. Therefore, we add another layer, called triple refinement, which prompts the LLM again to further decompose a triple with its object slot empty. The algorithm ends with the entity existence check, ensuring that the subject and object of all triples can all map to the entity nodes we created in the last stage.

The input and output of this stage are shown in Figure 8.2, with the input components in grey and the output components in colour.

The algorithm is as follows:

8 Stage III: Local Relation Extraction

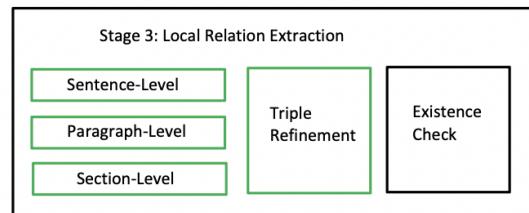


Figure 8.1: Stage 3: Local Relation Extraction

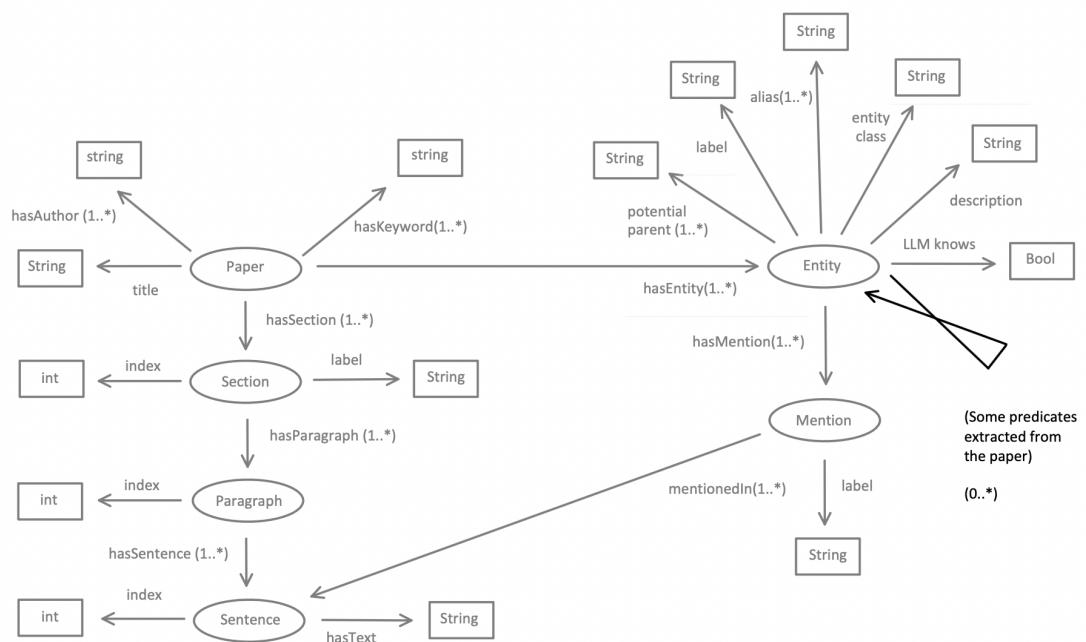


Figure 8.2: Input and Output

Algorithm

Input: An intermediate KG with entity nodes created.
Output: An intermediate KG with the addition of some predicates connecting between its entity nodes.

Creating inverse references from sentence/paragraph/section to entity.

For each entity in the KG,

for each mention of the entity,

find the sentence where the mentions refer to.

Put the entity IRI and the entity's local name (the mention's label) into a set under the sentence, as well as the set under the paragraph and section where the entity is mentioned.

Local Relation Extraction

For each section in the paper,

for each paragraph in the section,

for each sentence in the paragraph,

given a set of (entity_IRI, local_name) pairs, try to create an inverse mapping between the local names and entity IRI.

If there exist multiple identical local names referring to different entity IRI,

Modify the local to make it unique by adding some numbers, e.g., (1)

[LLaMA Prompt_1] Given the sentence and all the local names of all entities that the sentence contains, ask the LLM to extract the relations among them. Return the result in the form of a list of triples, where the subject and object must be chosen from the given list of local names of entities.

Store the triple under the sentence temporarily.

[LLaMA Prompt_1] Similarly.

[LLaMA Prompt_1] Similarly.

8 Stage III: Local Relation Extraction

```
# Triple Refinement

For each section in the paper,
    for each paragraph in the section,
        for each sentence in the paragraph,
            for each triple in the sentence,
                if its object is empty,
                    [LLaMA Prompt_2] Given the problematic triple, ask the
                    LLM to further decompose it into (Subject, Verb,
                    Predicate).

# Triple Existence Check

For each section in the paper,
    for each paragraph in the section,
        for each sentence in the paragraph,
            by using the mapping created before, find the actual entity
            IRI of the subject and the object of the triple.

            # This turn a triple of (local_name, predicate, local_name)
            into a triple of (IRI, predicate, IRI)

            If either the subject name or the subject name has no
            corresponding IRI,
                discard the triple.
            Else,
                move the triple from the sentence level to the paper level.

            Similarly.

            Similarly.

            Remove any redundant triples (triples with the same subject, predicate,
            and object) at the paper level.
```

8.2 Prompts

The following are the simplified versions of the three LLM prompts mentioned in the algorithm.

Prompt 1: Local Relation Extraction

```
## Task Definition
```

You are a linguistic expert involved in a relation extraction task for an academic paper. Given a text in the paper and a list of terms appearing in the text, your task is to extract the relations between these **terms** based on the context.

```
## Output Format ...
```

```
## Examples ...
```

```
## Your Task ...
```

Prompt 2: Entity Refinement

```
## Task Definition
```

You are a linguistic expert whose job is to decompose (S,VO) pairs into (S,V,O) triples, i.e. (Subject, Verb, Object) triples. Write your answer in JSON format.

```
## Examples ...
```

```
## Your Task ...
```

8.3 Assumptions Made

We assume that the relations, in the form of triples, returned by the LLM are indeed the relations that exist in the paper rather than something hallucinated by the LLM. We made this assumption because, unlike checking the existence of the entity, which can simply be performed via string matching or fuzzy matching, it is difficult to check the existence of a relation as well as the predicate in the relation. This is because we allow the predicate returned by the LLM to not necessarily syntactically appear in the document in order for the LLM to perform relation extraction freely to extract more relations. For example, the original text does not need to explicitly say that “ANU is located in Canberra” for the relation “(ANU, located in, Canberra)” to be extracted. This is because there are many ways to implicitly and semantically express such relations in a text. If we simply ensure all predicates must be in the original text, we will lose plenty of relations.

Due to this openness, there is also a possibility that the relation is hallucinated by the LLM. However, we assume that it is rare.

8.4 Limitations

Although we do not perform the existence check about the predicate of a relation, our algorithm still makes sure that the subject and object of a triple should at least be from the entities extracted before.

However, due to the stochastic behaviour of LLMs, the subject and object of triple returned sometimes do not have any matching entities. The subject and object can be the following three cases:

- It does not exist in the original text, neither syntactically nor semantically.
- It does not syntactically exist in the original text.
- It exists syntactically in the original, but it is missed by Stage 2: Entity Extraction of the pipeline.

To solve case three, we may add it to the graph as a new entity, and we may perform Stage 2: Coreference Resolution and Entity Disambiguation again. However, since there are multiple cases in which an extracted triple has a subject or object that does not match any existing entity in the graph, our pipeline chooses to discard the triple for simplicity, which leads to this limitation.

8.5 Complexity Analysis

The pipeline involves scanning through the paper 3 times at different hierachal levels. Therefore, the complexity involves $O(3L)$. After the initial extraction, we call the LLM again for any triple with an empty object. This is the upper bound by the number of triples extracted in the worst case. Therefore it is $O(R_1)$, where R_1 is the total number of relation (triples) so far. Since the number of relations can still grow in Stage 4: Location Relation Extraction, we use R_1 to only refer to the relations extracted in Stage 3, the current stage.

8.6 Decisions Made and Analysis

The major decision made was to use the local names of the entities in the prompt of asking the LLM to extract relations between the entities within a text. Using local names ensures that the entities provided in the prompt align with the provided text, which prevents the LLM from confusing. An entity can have different forms, and not all its forms appear in the text from which we want to extract the relations. The challenge is that the same local name may map to different entities. Therefore, disambiguation is required so that when an LLM outputs a triple consisting of two local names and one predicate, we can know the actual entities the local names refer to.

The second decision is about hierarchical extraction. This reason is similar to hierarchical extraction for entities, which allows the LLM to both have a specific and broad view of

8.6 Decisions Made and Analysis

a text, potentially making more relations extracted.

Chapter 9

Stage IV: Global Relation Extraction

This chapter will uncover the implementation details of Stage 5: Global Relation Extraction in 6 aspects, similar to the previous chapters, i.e., the algorithms, prompts, assumptions made, limitations, complexity, and decisions made.

9.1 Algorithm

The general pattern for global relation extraction is to perform a pairwise check of any possible pair of entities and see if there is a potential relation between them, given the context of the whole document. However, checking every possible pair can be computationally expensive. Therefore, our algorithm only selects the top 10% relevant entities in the paper. Even so, $O((0.1 \cdot n)^2)$ can still scale badly for long paper. Therefore, we set the maximum number of entities involved in global relation extraction to be 20.

To obtain the relevancy of each entity so that they can be ranked, the algorithm compares the embedding between the entity and the paper.

Because a whole paper can potentially exceed the context limit of the LLM for both encoders and decoders, the algorithm first shrinks the document based on the assumption that the first parts and the last parts of a paper are always the most important parts. The goal of shrinking is to keep the document as original as possible while the document can fit the LLMs. Therefore, the algorithm prompts the LLM decoder to do summarisations starting from its middle sections and up to the abstract and conclusion. If it turns out that the updated paper is still too long, another iteration will be performed.

Since the updated paper can now fit by an LLM encoder or encoder, the embedding of the paper can be obtained by calling an LLM encoder. We have chosen the encoder M3 instead of BGE due to a large context limit of 8196 tokens. However, we used M3 for coreference resolution because BEG specialised in generating embedding of English text

9 Stage IV: Global Relation Extraction

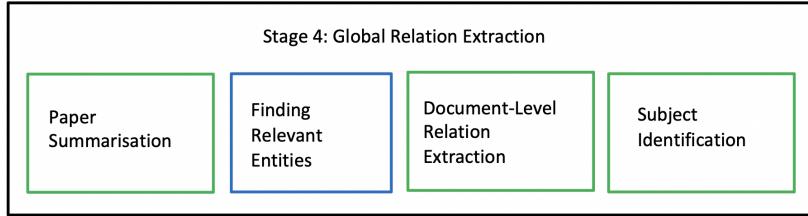


Figure 9.1: Stage 4: Global Relation Extraction

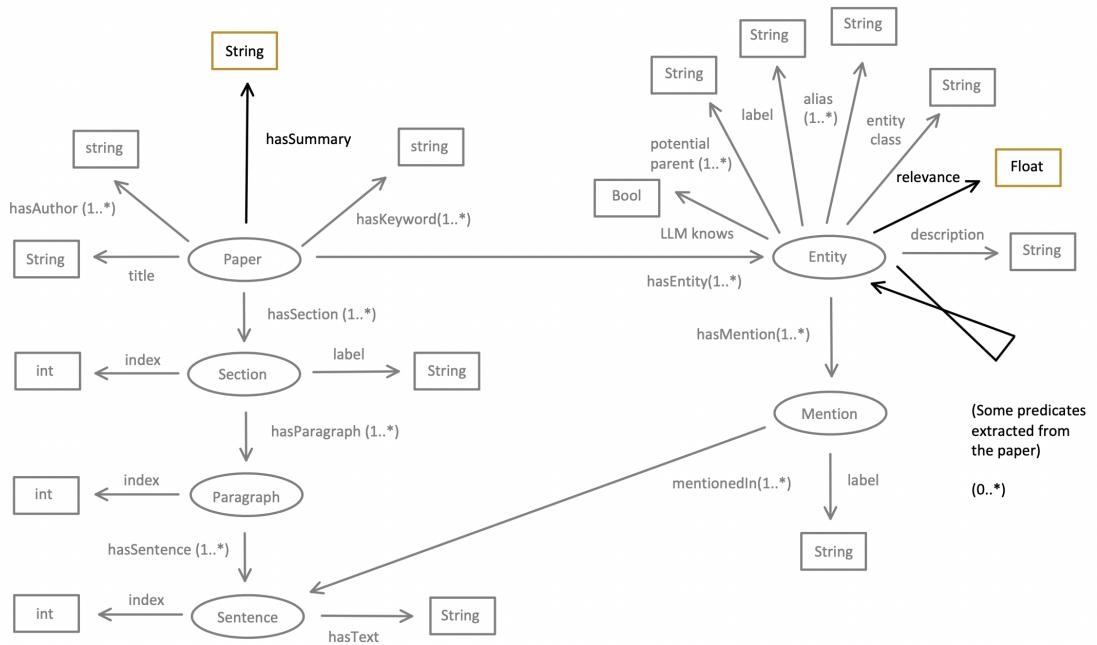


Figure 9.2: Input and Output

while M3 is multilingual, and our datasets are all in English.

For the substage of relation extraction, the algorithm loops through every possible pair within the most relevant entities. Within each iteration, the label, potential parent types, and description are given to the LLM. The paper's title, keywords, authors, and updated content with shorter length are also given. The output from the LLM is the most relevant predicate between them.

However, in practice, we found that even though the LLM can output a predicate between two entities, it cannot correctly identify the subject and object within the relation. Therefore, we perform another set of LLM calls, which give the label, potential parent types and description of both entities as well as their predicate. The LLM needs to identify the subject based on the given information.

9.1 Algorithm

The general picture of Stage 4 is shown in Figure 9.1, while the input and output are shown in Figure 9.2. The input is marked in grey, and the output is marked in colour. The algorithm of Stage 4 is as follows:

Algorithm

Input: An intermediate KG with some predicates connecting the entities.
Output: An intermediate KG with more predicates connecting the entities.

Paper Summarisation

While the paper's length is over the same token limit,

from the middle section to the first and the last section of the paper,

[LLaMA Prompt_1] Ask the LLM to summarise the section into half its original length.

If the updated paper is below the token limit,

break.

Finding Relevant Entities

[M3] Embed all the entities based on their label, potential parent types, and description.

[M3] Embed the updated paper

For each entity in the KG,

Compute the similarity score between the entity and the paper.

Store it as a new attribute under the entity.

Get the top 10% of the most relevant entities or the top 20 relevant entities from the KG into the list called relevant_entities.

Global Relation Extraction

For entity_1 in relevant_entities,

for entity_2 in relevant_entities,

[LLaMA Prompt_2] Using the paper's title, authors, keywords, and updated content, along with the label, description, and potential

9 Stage IV: Global Relation Extraction

parent types of both entities, ask the LLM to return at most one predicate between these two entities.

Subject Identification

For the newly extracted predicate of each relevant entity pair,

[LLaMA Prompt_3] Using the label, description, and potential parent types of both entities, plus the previously identified predicate, ask the LLM to determine the subject in this relation.

Tidy Up

Move all the triples to the paper level.

Return all redundant triples whose subject, predicate, and object are all the same.

9.2 Prompts

The following are the simplified versions of the three LLM prompts mentioned in the algorithm.

Prompt 1: Paper Summarisation

Task

You are a linguistic expert involved in a text summarisation task. Given the text in the **Input** section below, your job is to summarise it into half its original length while retaining key information and ideas.

Input

Output

Prompt 2: Global Relation Extraction

Task Definition

You are a linguistic expert involved in a relation extraction task for finding the relationship between the terms "{term_1}" and "{term_2}" in an academic paper. Given the title, authors, keywords, and content of the paper as well as the types and description of each term, your task is to extract the **most important** predicate that connects the term "{term_1}" and the term "{term_2}".

Output Format ...

```
### Paper
**Title**: {title}
**Keywords**: {keywords}
**Authors**: {authors}
**Content**: {content}

### Term 1: {term_1} ...
### Term 2: {term_2} ...
**Output**:
```

Prompt 3: Subject Identification

Task Definition

You are a linguistic expert involved in a subject identification task. Given the terms "{subject_label}" and "{object_label}" related by the predicate "{predicate}", your task is to determine the subject of this relation. Your answer should be a triple, providing the new orders for Term 1 and Term 2.

- Your answer should be based on the labels, types, and descriptions of both terms, the predicate, and your common knowledge.
- You should also check if the predicate is active or passive in order to determine the subject
- Ask yourself whether "{subject_label} {predicate} {object_label}" is more fluent or "{object_label} {predicate} {subject_label}" is more fluent and natural

```
## Output Format ...
## Input ...
## Output ...
```

9.3 Assumptions, Decision Made, and Limitations

During the implementation, we have assumed that the relations returned by the LLM are indeed the relations that exist in the document.

However, it was found that that was not always the case. The LLM we used, LLaMA, suffered seriously from hallucination issues in this stage. That is, for each relevant entity pair, no matter if they are truly related by some predicate or not, the LLM always return predicates, even though we have explicitly stated in the prompt that “at most one” predicate should be returned.

In fact, in the original implementation, we even do not have this “at most one” constraint. That is, we allow the LLM to return any number of predicates found in the document.

9 Stage IV: Global Relation Extraction

However, the finding is that only the first one or two predicates are usually the correct predicate in between the entities, while others are merely "filled in the numbers".

Therefore, we made the decision that only one predicate is allowed to be returned between any two entities. This decision allows the LLM to focus on returning the most likely predicate.

In the original implementation, there is also another layer, which, given the triple found during the global relation extract, asks the LLM again to check whether the relation between two entities exists in the paper. However, it turns out that even a simple 'Yes' or 'No' response can itself be a hallucination.

Therefore, we decided not to include such a layer to reduce the overall complexity. The method for reducing hallucination is based on restricting the number of predicate returns. And we assume that the only predicate return is truly a predicate in the paper.

Since there is only one predicate return, this leads to the limitation that if there indeed exists multiple predicates between some entities, the pipeline will miss most of them.

9.4 Complexity Analysis

The complexity of this stage can be modelled by $O(L + \text{Max}(20^2, (0.1 \cdot E)^2) + R_2)$, where L is the length of the paper, E is the number of entities, and R_2 is the number of triples extracted in Stage 4, the current stage only. During the summarisation, our algorithm loops through each section of the document and summarises each section individually. However, there is not an upper bound on how many times it loops through the whole paper. Since academic papers usually have a fixed length restricted by some academic conference and it does not go extremely long, it turns out that the summarisation can usually finish within one epoch (all the sections in the paper). Therefore, we model it as $O(L)$ for simplicity. For the global relation extraction part, this is either the top 20 entities or the top 10% entities. Therefore, it is $O(\text{Max}(20^2, (0.1 \cdot E)^2))$ since pairwise comparison is formed. In the end, for each triple found, the algorithm further confirms the subject and object, leading to an additional complexity of $O(R_2)$.

Chapter 10

Stage V: Schema Generation

This chapter will provide the implementation details about the second last stage, schema generation. It will also follow a similar layout to the previous chapter, i.e., the algorithms, prompts, assumptions made, limitations, complexity, and decisions made. This is also the last chapter about pipeline implementation because the last stage, KG post-processing, has been covered in Chapter 5, the pipeline overview.

10.1 Algorithm

There are two main parts for creating a concrete schema of the KG. One is to construct a taxonomy for the entities within the KG. The other is to normalise predicates.

The first part creates a hierarchy of supertype and subtype among all entities in the KG, e.g., (“ANU”, “has a broader term”, “Public University”) and (“Public University”, “has a broader term”, “University”) as long as all “ANU”, “Public University”, and “University” are the entity exist in the KG, which essential form a directed acyclic graph. In fact, some of the relationships may have already been found in Stage 3: Local Relation Extraction and Stage 4: Global Relation Extraction. However, this stage aims to make the KG more complete by only focusing on the hierarchical relations.

The other is to merge predicates referring to the same meaning so the redundancy of predicates can be reduced, which can further reduce the number of redundant triples, like (“ANU”, “located in”, “Canberra”) and (“ANU”, “is in”, “Canberra”).

The high-level pattern of the algorithm of this stage is similar to Stage 2: Entity Disambiguation and Coreference Resolution. In the taxonomy part, the essential goal is to find a matching parent entity in the KG for each potential parent of the entities so that an edge, “has a broader term” or “skos:broader” is drawn between an entity and its parent entity. Before this stage, the potential parent types are not entities but strings,

10 Stage V: Schema Generation

and this stage matches some of them to the actual entities in the KG.

The algorithm starts with gathering the potential parents of all entities into a Python Dictionary, where the key is the name of the potential parent. We made an assumption that the potential parent types are unambiguous. This is because entity types are usually general concepts, and the fact that they are general makes them unlikely to have too many distinct meanings. This is because their meanings are already broad. This assumption is essentially the same as one of the assumptions made during the Entity Disambiguation in Stage 2. With this parent-type dictionary. The algorithm prompts the LLM to generate a description of each type without any assistance context. Since they are general, we assume that the LLM knows all of them. After that, we embed these parent types using their labels and descriptions, as well as embedding all the entities in the KG. For each entity and for each potential parent type, the algorithm finds the embedding of the parent type by checking the parent-type dictionary. And compare it with all the other actual entities in the KG, if it turns out that a potential parent of an entity is highly similar to an entity on the KG according to their embedding, we create a “skos:broader” relationship between these two entities. After the taxonomy graph is created, we perform a basic consistency check by removing any loop in the directed acyclic graph of taxonomy using the Depth-First Search Algorithm ([Tarjan, 1972](#)).

As for predicate normalisation, this is highly similar to coreference resolution. However, we do not simply merge two predicates only based on their label, like when merging named entities and general concepts because the meaning of a predicate can vary a lot depending on the context, like (“Amy”, “run”, “a business”) and (“Amy”, “run”, “a marathon”). Therefore, we perform a more conservative approach. That is, first, generate a description for the predicate based on the triple, then embed the predicate and finally merge the predicate if their embeddings are found similar.

After the predicate normalisation, the number of predicates is fixed. Then we assign each predicate an IRI.

In the end, we eliminate any repeating triple after the normalisation where all its subject IRI, object IRI, predicate IRI are the same.

The general picture of Stage 4 is shown in Figure 10.1, while the input and output are shown in Figure 10.2. The input is marked in grey, and the output is marked in colour. The ontology of the KG does not change, but only more relations between entities are created, and the number of predicates is reduced.

The algorithm of Stage 4 is as follows:

Algorithm

Input: An intermediate KG with some predicates and some (entity, predicate, entity) triples

Output: An intermediate KG with less predicates and more

10.1 Algorithm

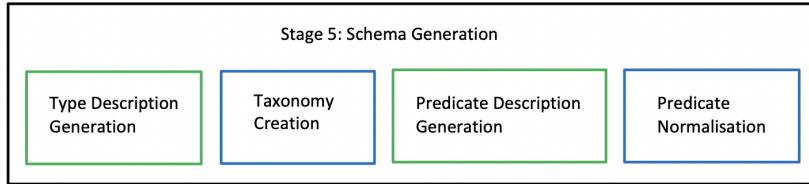


Figure 10.1: Stage 5: Schema Generation

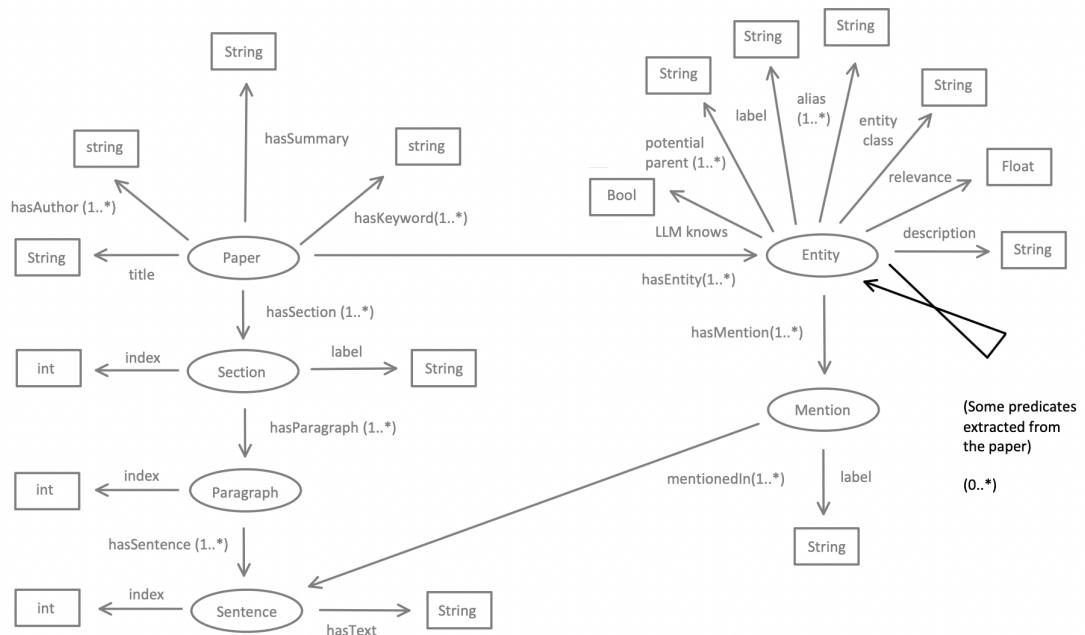


Figure 10.2: Input and Output

10 Stage V: Schema Generation

```
(entity, predicate, entity) triples

# Gathering Types

Create a dictionary for storing potential parent types for all entities,
named as type_dict.

For each entity in the KG,
    for each potential parent type of the entity,
        add it into the type_dict.

# Generating Type Description

For each type in type_dict,
    [LLaMA Prompt_1] asks the LLM to generate a general description based
    on its common knowledge.

# Embed Types and Entities

[BGE] Embed all types in type_dict based on their label and description
[BGE] Embed all entities in the KG based on their label and description

# Finding Parents

For entity_1 in the KG,
    for each potential parent type of the entity,
        get the embedding of the potential parent type from type_dict.

    For entity_2 in the KG,
        compute the cosine similarity between the embedding of the
        potential parent type and the embedding of entity_2
        if the similarity > 0.95,
            draw a temporary and directed edge between entity_1 and
            entity_2.
```

```
# Removing Cycles
```

For the subgraph created by the temporary links, use the Depth-First Search algorithm to remove all cycles.

Put the updated temporary links into the KG.

```
# Predicate Normalisation
```

Now the KG has a list (set) of (Entity IRI, predicate name, Entity IRI) triples.

```
# Generate Descriptions for Predicates
```

For each triple in the KG,

[LLaMA Prompt_2] Given the triple, ask the LLM to generate a general description for the predicates.

Store the description directly after the triple. (essential making it into 4-tuple)

```
# Embed Predicate
```

[BGE] Embed the predicate of all triples based on their labels.

Store the embedding directly after the triple. (essential making it into 5-tuple)

```
# Finding relevant predicates
```

For triple_1 in the KG,

for triple_2 in the KG,

get the embeddings of the predicates of triple_1 and triple_2.

Compute their cosine similarity.

If their cosine similarity is over 0.95,

draw a temporary link between these two triples.

10 Stage V: Schema Generation

```
# Predicate Resolution

For each triple in the KG,

    pull out their predicate into a graph, as well as the temporary links
    between these predicates.

    Maintain a mapping between the predicate in the triple and the node in
    the predicate graph.

# Up to now, all predicates are still distinct nodes.

Use the modified Bron-Kerbosch algorithm to find all
the disjoint maximal cliques in the predicate graph.

For each clique of predicates (including those with only one node),

    group them into a single predicate.

    Choose any of the names of the predicate as a representative name for
    the final predicate.

    Assign a unique IRI for the predicate.

For each triple in the KG,
    change it from (Entity IRI, predicate name, Entity IRI) into
    (Entity IRI, Predicate IRI, Entity IRI) using the mapping created
    before.

Remove any redundant predicates with the same IRIs in all positions.
```

10.2 Prompts

The following are the simplified versions of the three LLM prompts mentioned in the algorithm.

Prompt 1: Type Description Generation

```
## Task
```

```
You are a linguistic expert involved in a description
writing task for a term "{label}" found in a paper.
Given the term and based on your background knowledge
about the term, your task is to write a single-
sentence description that can precisely tell what the
actual entity or concept the term "{label}" refers to.
```

```
## Output Format ...
## Input ...
## Output ...
```

Prompt 2: Predicate Description Generation

```
## Task
You are a linguistic expert involved in writing a
general description for the predicate "{predicate}" in
the (subject, predicate, object) triples like "
({subject}, {predicate}, {object})".
```

- Your description should be general and not tied to any specific instance of the given triple.
- The provided triple is only there to help you understand the use of the predicate.
- Your description should explain how the predicate connects the subject and object in a general sense.

```
## Output Format ...
## Input ...
## Output ...
```

10.3 Assumptions Made

The following is the summary of the key assumptions made in this stage, some of which have been mentioned earlier.

1. When grouping the potential parent types into a dictionary based on their names, we assume that these types are ambiguous. That is, their labels are sufficient to represent themselves.
2. When generating descriptions for all potential parent types, we assume that the LLM has all background knowledge about the types, similar to generating descriptions of predicates.

10.4 Limitations

Several limitations can be found in this stage:

1. The potential parent types are created at the very first stage of the pipeline before coreference resolution. Therefore, these potential parent types are created based on the mention of entities within a limited context only. Therefore, they may not be highly accurate. A potential way to reduce this limitation is to type any entity after the Coreference Resolution stage. However, it may introduce extra complexity.

10 Stage V: Schema Generation

2. When generating descriptions for types and predicates, the description may not truly represent the types and predicates, i.e. they can be lossy.
3. When generating embedding for types and predicates, the embedding may not truly represent the types and predicates, i.e. they can be lossy. A more advanced technique for generating embedding for building taxonomy is to create box embedding ([Onoe et al., 2021](#)) embedding for entities in the KG. In this way, if the embedding of one entity is involved in the box embedding, then we can know that one is the parent entity of the other.
4. When determining the parent entity of an entity and checking if the two predicates are similar, we use the cosine similarity between their embedding. However, the cosine similarity does not tell the length of these two vectors. Therefore the result can be lossy.
5. After computing the cosine similarity, we create a temporary link between the entity and its parent entity or between two predicates as long as their similarity is above some threshold with further double-checking, unlike the Coreference Resolution stage. This is because we tend to improve the efficiency of the pipeline. However, such a more aggressive way of linking entities or predicates can lead to more errors.
6. When merging a group of predicates into a single predicate, the algorithm does not have an extra step and heuristic to find the most representative name for the predicates for simplicity.

10.5 Complexity Analysis

The complexity of this stage can be modelled as $O(E + R)$, where E is the number of entities and R is the total number of predicates (= triples) of the relations of the entities after the local and global relation extraction. Although an entity normally has several potential entity types, some potential entity types are shared among different entities, e.g., the “ANU” and “USYD” are all “Universities”. Therefore, for simplicity, we roughly model it as $O(E)$. Since the algorithm also call the LLM decoder to generate a description for the predicate of each triple, there is also a complexity of $O(E)$, which results in an overall complexity of $O(E + R)$.

10.6 Design Decision

Some decisions made in this stage are similar to Stage 3: Entity Disambiguation and Coreference Resolution. They include the use of description for generating embedding and the use of disjoint maximal cliques to group relevant predicates.

However, the major difference is the removal of the double-checking mechanism when two of the potential parents of an entity are found similar to another entity, or two

10.6 Design Decision

predicates are found similar. This decision may reduce complexity but potentially result in lower precision. However, the recall may also increase because the double-checking mechanism itself is error-prone, and LLM can be too conservative. Therefore, this is a trade-off. Eventually, we increase the threshold from 0.9 to 0.95 to prioritise precision over recall.

Chapter 11

Evaluation

This chapter will evaluate the performance of our KGC pipeline through a number of different approaches.

All results, as well as the tools we created for evaluation, are available at <https://gitlab.anu.edu.au/u1085404/anu-scholarly-kg/-/tree/master/llm-based/LLM-KGC-v0-Tests>

11.1 Experiment Setup

To enable all different evaluations, the starting point is to use our KGC pipeline to build KGs. The test sets we have used to build KGs are as follows:

11.1.1 Dataset 1: Five Full Papers

This set consists of 4 full papers from ANU ASKG plus 1 full paper not in ANU ASKG.

The 4 papers from ANU ASKG are already semi-structured, i.e., consisting of a hierarchy of sections, paragraphs, and sentences, and they are in the TTL formats, which can be directly accepted by our KGC pipeline. The ones outside of ASKG are in raw PDF format. Therefore we performed a manual decomposition into a similar hierarchy structure as those in ASKG and stored the result in JSON, which can then also be accepted by our pipeline by skipping the initial TTL-to-JSON stage.

Specifically, the title of each paper is:

1. MEL: Metadata Extractor & Loader ([Rodríguez Méndez et al., 2021](#))
2. Modeling Actuations in BCI-O: A Context-based Integration of SOSA and IoT-O ([Rodríguez Méndez, 2018](#))

11 Evaluation

3. Building An Open Source Linux Computing System On RISC-V ([Ince et al., 2019](#))
4. TNNT: The Named Entity Recognition Toolkit ([Seneviratne et al., 2021](#))
5. A Pipeline For Analysing Grant Applications ([Pan et al., 2022](#))

11.1.2 Dataset 2: SciERC

The dataset SciERC ([Luan et al., 2018](#)) contains the mappings between 500 semi-structured abstract-only academic papers and their corresponding KGs. We regard them as semi-structured because, as in most NLP datasets, the papers have been preprocessed and sentence-tokenised, stored in JSONs. Although the target KGs are supplied, as mentioned before, these KGs are considered the results of traditional KGC approaches with a fixed and predefined number of types and predicates. However, they are still valuable references during our manual evaluation in Section [11.5](#).

The dataset had already been split into the training set (350), test set (100), and development set (50) when we downloaded it. Therefore, we ran our pipeline on their given test set consisting of 100 abstract-only papers.

Although they are already semi-structured, a basic preprocessing was done to align their structure with those in ASKG so that they match the interface of our pipeline.

11.1.3 Platform

The platform to create KGs from the datasets and perform the later evaluation processes is a desktop with the following specifications.

- CPU: AMD Ryzen 5 5600X 6-Core Processor
- Memory: 16 GB
- GPU: GeForce RTX 3080 12GB
- OS: Pop! OS 22.04 LTS

11.1.4 LLMs

Three of the four LLMs used are the same as the development process. However, we also used GPT-4o-Mini as the decoder of our pipeline to create KGs in order to see the portability and compatibility of our pipeline with different LLMs. Although we have listed two encoders, they are not the variable in our testing as they are used in different stages for different purposes, as mentioned before. Only LLaMA and GPT are our variables and are switched during the testing.

Decoder 1:

- **Named:** Meta-Llama-3-8B-Instruct.Q4_0.gguf
- **Size:** 4.66GB

- **Context Limit:** 8192
- **Availability:** <https://huggingface.co/QuantFactory/Meta-Llama-3-8B-Instruction-GGUF/tree/main>

Decoder 2:

- **Named:** gpt-4o-mini
- **Size:** Unknown
- **Context Limit:** 128k
- **Availability:** <https://platform.openai.com/docs/models/o1>

Encoder 1:

- **Named:** BAAI--bge-base-en-v1.5
- **Size:** 438.9MB
- **Context Limit:** 512
- **Availability:** <https://huggingface.co/BAAI/bge-base-en>

Encoder 2:

- **Named:** BAAI--bge-m3
- **Size:** 2.29GB
- **Context Limit:** 8192
- **Availability:** <https://huggingface.co/BAAI/bge-m3>

11.2 General Evaluation

With 100 abstract-only papers plus five full papers with two different LLM decoders, we created 204 different KGs $[(100 + 5) * 2 = 210]$. The tables below show the general information about these KGs and the runtimes during the construction.

- **Pipeline Complexity:** Table 11.1
- **Five-Full-Papers + LLaMA:** Tables 11.2 and 11.3
- **Five-Full-Papers + GPT:** Table 11.4 and 11.5
- **SciERC + LLaMA:** Table 11.6 and 11.7
- **SciERC + GPT:** Table 11.8 and 11.9

The complexity table (Table 11.1) shows the theoretical complexity of each stage of the pipeline. They have been derived from the previous chapters (Chapters 6 to 10). The complexity is measured by the number of times when an LLM decoder is called. That is,

11 Evaluation

for a single LLM decoder call, it is $O(1)$. Since a single LLM decoder call is dominantly expensive, the complexity of all the other computations, e.g., the algorithmic processes, file IOs, and LLM decoder calls, can all be ignored. The meaning of each symbol in the complexity analysis is as follows:

- L : The length of the paper
- E : The number of entities
- M : The number of mentions
- R : The number of relations (triples between any two entities)

For the rest of the tables, the meaning of each column is as follows:

- **Length**: The length of the paper, measured in tokens.
- **Entities**: The total number of entities nodes in the output KG.
- **Mentions**: The total number of mentions nodes in the output KG. Each entity has at least one mention. Different mentions may correspond to the same entity.
- **Relations**: The number of relations (triples between any two entities) in the output KG.
- **Isolated Entities**: The number of entities that are not related to any other entities in the output KG. That is, there is no outward edge from these entity nodes except those pointing to their attributes, e.g., their labels, aliases, and descriptions. Commonly speaking, the fewer, the better.

The actual complexity of KGC using LLaMA is measured by time (minute). We have found in the Background Chapter (2.4.5) that the complexity of the LLM decoder is at least quadratic and potentially cubic. However, since each prompt our pipeline made is approximately the same length, and thus each LLM decoder call should take roughly the same constant amount of time, if we treat the time taken by each LLM decoder call as $O(1)$, the overall runtime should align with the theoretical complexity shown in Table 11.1.

The actual complexity of KGC using GPT is measured by cost (USD). Since each LLM call is done through the OpenAI API, and the network speed and the server reaction time can be very unstable, we did not use time as our measurement. However, we found that the cost for each LLM call through the OpenAI API is about linearly proportional to the length of each prompt. Since each prompt our pipeline made is approximately the same length, and thus each prompt should have a similar cost, if we treat the cost by each LLM decoder call as $O(1)$, the overall cost should also align with the theoretical complexity shown in Table 11.1.

Table 11.1: **Complexity** of the Pipeline

Stage	Complexity
0	$O(0)$
1	$O(9L)$
2	$O(2M + M^2)$
3	$O(3L + R_1)$
4	$O(L + \text{Max}(20^2, (0.1 \cdot E)^2) + R_2)$
5	$O(E + R)$
6	$O(0)$

Table 11.2: **Statistics** of the KGs constructed from the **Five-Full-Papers** dataset by **LLaMA**

Paper	Length (Token)	Entities	Mentions	Relations	Isolated Entities
1	1165	147	243	235	38
2	3161	351	747	860	73
3	3062	473	753	894	106
4	1486	201	339	518	41
5	5183	653	1329	1437	113
Mean/Std	2811 ± 1434	356 ± 184	682 ± 384	789 ± 404	74 ± 31

Table 11.3: **Runtimes** (in Minutes) of KGC from the **Five-Full-Papers** dataset by **LLaMA**

Paper	S0	S1	S2	S3	S4	S5	S6	Time
1	1.466E-6	6.336	20.18	3.461	12.58	4.775	2.078E-3	47.34
2	2.587E-6	18.75	117.7	9.636	38.59	13.28	3.851E-3	198.0
3	5.126E-6	20.36	79.03	11.47	24.09	12.65	4.214E-3	147.6
4	3.719E-6	8.152	42.93	4.585	29.84	7.782	2.089E-3	93.29
5	3.703E-6	29.72	354.6	16.22	37.43	16.79	6.411E-3	454.8
Mean/Std	3.320E-6 ± 1.228E-6	16.66 8.574	122.9 120.5	9.074 4.665	28.51 9.551	11.06 4.256	3.729E-3 ± 1.604E-3	188.2 142.6

11 Evaluation

Table 11.4: **Statistics** of the KGs constructed from the **Five-Full-Papers** dataset by **GPT**

Paper Index	Length (Token)	Entities	Mentions	Relations	Isolated Entities
1	1165	180	301	526	14
2	3161	313	743	1130	24
3	3062	473	803	1313	45
4	1486	209	395	837	18
5	5183	638	1444	1954	47
Mean/Std	2811 ± 1434	363 ± 172	737 ± 403	1152 ± 482	30 ± 14

Table 11.5: **Costs** (in USD) of KGC from the **Five-Full-Papers** dataset by **LLaMA**

Paper	Cost (USD)
1	0.21
2	0.62
3	0.53
4	0.32
5	1.45
Mean/Std	0.63 ± 0.44

Table 11.6: **Statistics** of the KGs constructed from the **SciERC** dataset by **LLaMA**

Paper	Length (Token)	Entities	Mentions	Relations	Isolated Entities
Mean/Std	134 ± 51	29 ± 10	37 ± 13	19 ± 11	10 ± 5

Table 11.7: **Runtimes** (in Minutes) of KGC from the **SciERC** dataset by **LLaMA**

Paper	S0	S1	S2	S3	S4	S5	S6	Time
Mean/Std	9.119	9.989		5.834	2.989	9.623	2.059	
	E-6	E-1	2.110	E-1	E-1	E-1	E-4	4.954
	±	±	±	±	±	±	±	±
	1.731	2.568	1.078	2.014	2.125	3.099	1.346	1.940
	E-6	E-1		E-1	E-1	E-1	E-4	

Table 11.8: **Statistics** of the KGs constructed from the **SciERC** dataset by **GPT**

Paper Index	Length (Token)	Entities	Mentions	Relations	Isolated Entities
Mean/Std	134 ± 51	28 ± 10	37 ± 15	33 ± 18	4 ± 2

Table 11.9: **Costs** (in USD) of KGC from the **SciERC** dataset by **GPT**

Paper	Cost (USD)
Mean	0.02

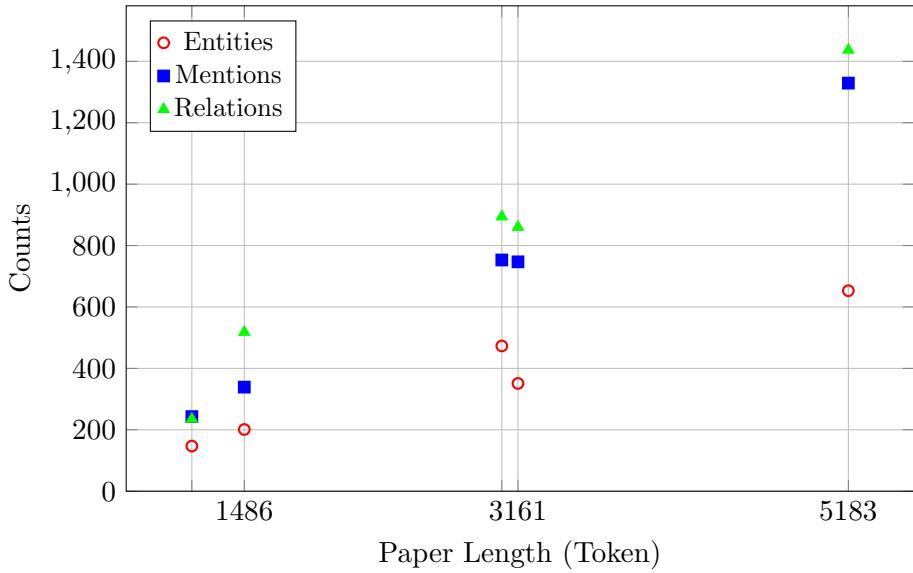


Figure 11.1: **Statistics** of the KG constructed from the **Five-Full-Paper** dataset by **LLaMA** vs. Paper Length

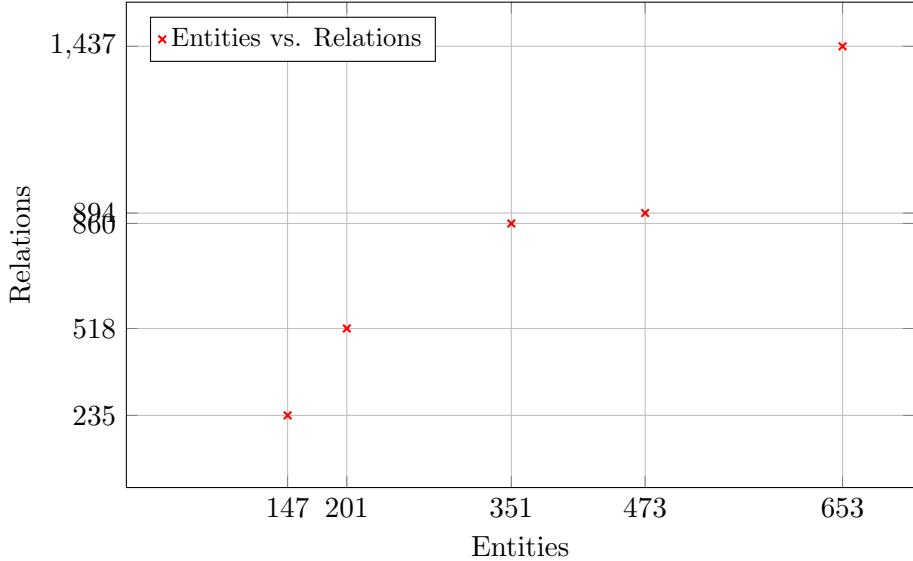


Figure 11.2: **Entities vs Relations** of the KG constructed from the **Five-Full-Paper** dataset by **LLaMA**

11 Evaluation

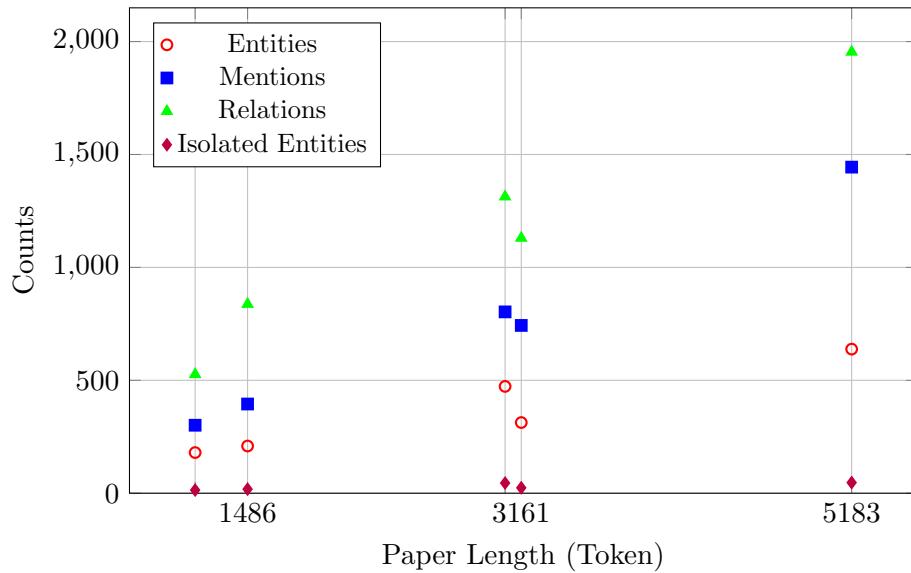


Figure 11.3: **Statistics** of the KG constructed from the **Five-Full-Paper** dataset by **GPT** vs. Paper Length

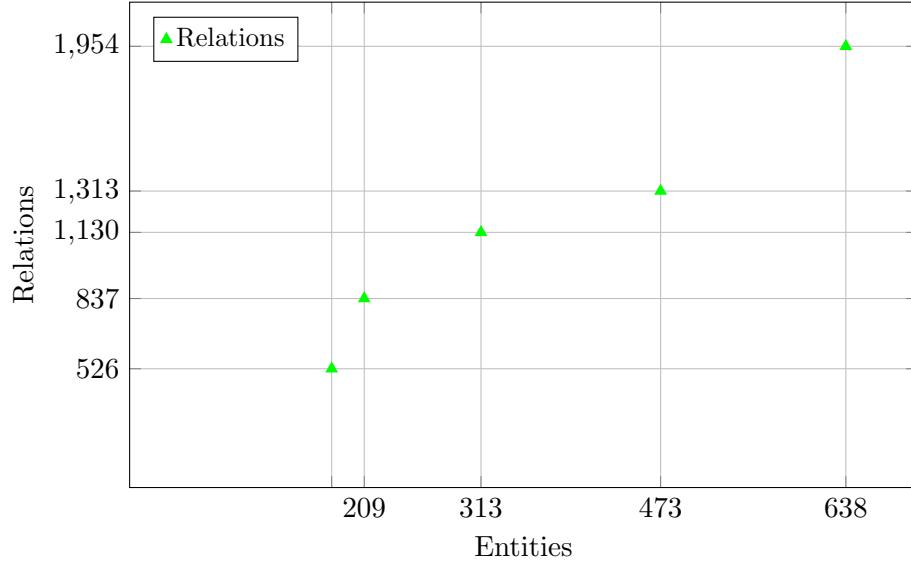


Figure 11.4: **Entities vs Relations** of the KG constructed from the **Five-Full-Paper** dataset by **GPT**

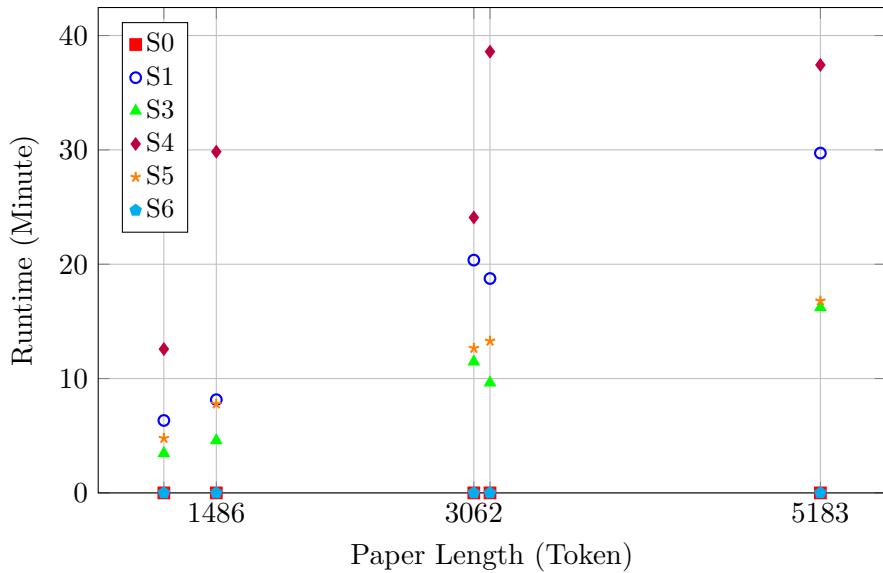


Figure 11.5: **Runtimes of Stages S0, S1, S3, S4, S5, and S6 of KGC from the Five-Full-Paper dataset by LLaMA vs. Paper Length**

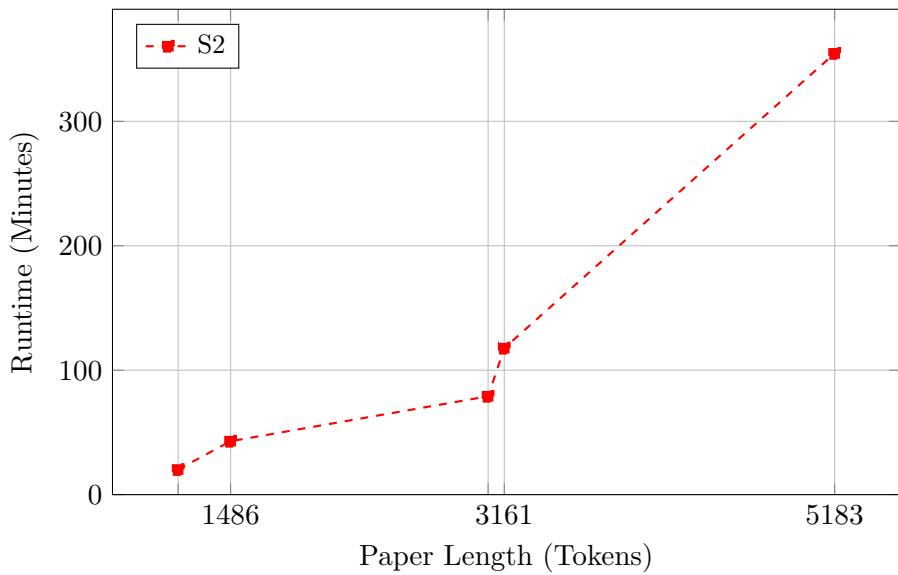


Figure 11.6: **Runtimes of S2 of KGC from the Five-Full-Paper dataset by LLaMA vs. Paper Length**

11 Evaluation

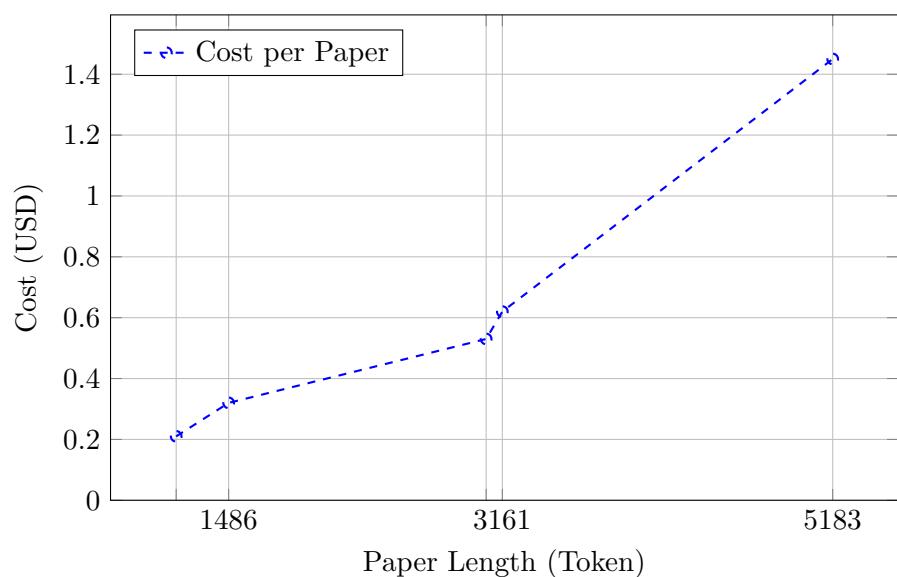


Figure 11.7: Costs of KGC from the **Five-Full-Papers** dataset by **GPT** vs. Paper Length

11.2.1 Discussion on the KGs

From the Figures 11.1, 11.2, 11.3, and 11.4 and Tables 11.2, 11.4, 11.6 and 11.8, some key observations can be drawn.

1. For both datasets and for both LLM decoder models (Tables 11.2, 11.4, 11.6, and 11.8), the pipeline is *capable* of extracting some entities and relations, although we cannot measure the quality of these entities and relations at the moment.
 - a) The pipeline based on GPT seems to be *more capable* of finding relations between entities as the resultant KGs have statistically lower * isolated entities. This makes sense because the specific version of LLaMA we used is considered lightweight compared to GPT running on servers. Similarly for the SciERC dataset shown in Tables 11.6 and 11.8.
2. The number of Mentions is approximately *linearly proportional* to the length of the paper. The number of Entities is slightly *sub-linearly proportional* to the length of the paper. This can be shown in Figures 11.1 and 11.3 for both KGs constructed by LLaMA and GPT. The number of relations is *linearly proportional* to the number of entities, as shown in Figures 11.2 and 11.4.
 - a) The *linear relation* between mention count and paper length is as expected because the longer the paper is, the more words or phrases it should have.
 - b) Commonly speaking, the expected relation between entity count and paper length is indeed *sub-linear*. This is because most papers should have a central theme rather than keep introducing new things (entities). Although the longer the paper is, the more mentions of these things are, the number of things should not grow as fast as their mentions. Therefore, this is as expected. However, since the relation is only slightly sub-linear, this suggests that the Coreference Resolution stage (Stage 3) may *not be performed thoroughly*, i.e., some entity mentions (that should be merged into the same entity as they refer to the same thing) are treated as two separate entities.
 - c) In a general sense, the expected number of Relations should grow *more than linearly proportionally* to the Entities. This is because a graph with more nodes can potentially have quadratically more edges. However, based on Figures 11.2 and 11.4, the number of relations seems to be linearly proportional to the number of entities. This suggests that there are potentially some relations *missed by the pipeline*.

* The Isolated Entities column of Table 11.2 [38, 73, 106, 41, 113] and the Isolated Entities column of Table 11.4 [14, 24, 45, 18, 47] have p-value 0.032

11.2.2 Discussion on the Complexity

From the Figures 11.5, 11.6 and 11.6 derived from Tables 11.2, 11.3, 11.4, 11.5, key observations can be drawn.

11 Evaluation

1. The runtimes of Stage 0 (KG Preprocessing) and Stage 6 (KG Postprocessing) are *approximately zero*, regardless of the length of the paper, as shown in Figure 11.5. This aligns with the $O(0)$ theoretical complexity of the pipeline as they do not involve LLM decoder calls.
2. The runtimes of Stage 1 (Entity Mention Extraction) and 3 (Local Relation Extraction) are approximately *linearly proportional* to the length of the paper. This also aligns with the theoretical results, $O(3L)$ and $O(L + R)$.
3. The runtimes of Stage 4 (Global Relation Extraction) and the length of the paper are *not quite correlated*. However, it is bounded by a constant, which is as expected because we have set the maximum of entities to compare with each other to 20 to prevent $0.1N$ from growing quadratically.
4. The runtime of Stage 5 (Schema Generation) tends to be *sub-linearly* related to the length of the paper. This is as expected because, as discussed previously (11.2.1), the measured relation between Entities and paper length is sub-linear, and the measured relationship between Relations and Entities is linear. Since the theoretical complexity of Stage 5 is $O(E + R)$ and both E and R are sub-linearly proportional to L , the measured runtime is, therefore, sub-linear.
5. The relationship between the runtime of Stage 2 (Coreference Resolution and Entity Disambiguation) is clearly more than linear, as shown in Figure 11.6. This aligns with the $O(2M + M^2)$ complexity, and since M is linearly proportional to L . In Stage 2, our pipeline filters out most of the unrelated mentions based on their embedding and only calls the LLM decoder whenever they are highly similar. Even with blocking, the nature of this substage is still a pairwise comparison, which can make the complexity of this stage scale very quickly in the worst case. Table 11.3 shows that the runtimes of all papers are dominant by this single stage, leading to non-linear growth of runtime (Table 11.3) or cost (Figure 11.7) with respect to the length of the paper. Therefore, it is worth checking if all components in Stage 4 are essential and if there are any possibilities to simplify Stage 4 and increase the pipeline performance without losing too much quality of the constructed KG. One solution is that whenever two entity mentions are similar in their embedding, the pipeline directly merges them into the same entities without double-checking with the LLM decoder. In this case, the expected complexity can be reduced to $O(2M)$. This led to an ablation study in Section 11.6.

11.3 Evaluation via Reverse Engineering

The last part of the evaluation mainly focuses on the shallow features of the KGs. They are important but not sufficient. Starting from this section, the evaluation will be directly based on the quality of the constructed KGs.

The main challenge of evaluating the quality of the KGs is that there are no references,

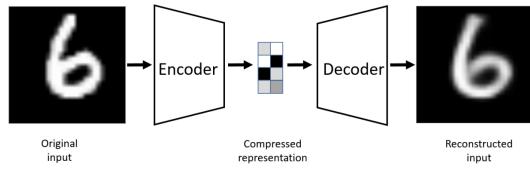


Figure 11.8: Structure of an Auto-encoder (Bank et al., 2020)

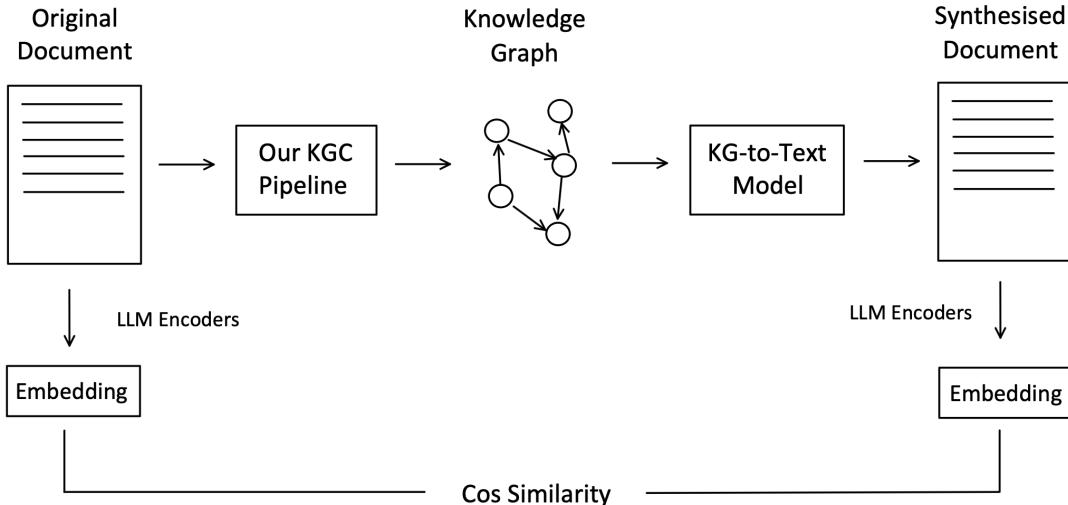


Figure 11.9: Evaluation via Reverse Engineering

i.e., the datasets are unlabelled. Even though SciERC is labelled, it does not contain full papers and targeted KGs in the dataset are considered KGs constructed through traditional KGC.

The first evaluation approach is inspired by how auto-encoder models are trained and tested.

In general, an autoencoder takes a source as input and converts it into some compressed representation through its encoder, also referred to as the feature extractor. After that, it tries to reverse the compressed representation back to the original source. By comparing the original source and the synthesised source, we can then know how good the feature extractors and decoder are. This is because a good auto-encoder should capture most of the key features of the input source so that it is possible to recover the source back from the key features with minimal loss. (Bank et al., 2020)

Similarly, our KGC pipeline can be viewed as the encoder part of an autoencoder, whereas the compressed representation is KG. Therefore, if we have a model to convert KGs back to text documents, we can then evaluate our pipeline and KG-to-Text model together based on the similarity between the original documents and the synthesised

11 Evaluation

documents. A common way to measure the similarity between two documents is by comparing the cosine similarity of their embeddings, as embeddings also represent the key features of the two documents.

In fact, KG-to-Text is another important field of study that is parallel with KGC in the broader field of KG (Pan et al., 2023). Because the study of KG-to-Text can be significantly deep, we only implemented a straightforward approach to convert our KGs back to text documents.

11.3.1 Evaluation Approach

This approach is also considered the baseline approach in many KG-to-Text studies, as illustrated in Pan et al. (2023)'s survey article. That is, we use an LLM decoder to concat all (Subject, Predicate, Object) triples of Entities in the constructed KG.

However, as a minor extension to the baseline, we process the triples in batches, and the most relevant triples are processed first, which ensures the most important knowledge of the document has been conveyed at the beginning, which aligns with the nature of academic writing. When concatenating triples, we also supply the LLM with the description of entities involved in the triples. The relevance of triple is calculated by the sum of the relevance score of its subject and object entities. The relevance scores of entities are calculated by the cosine similarity between the embedding of the entities and documents, which is already done in Stage 8: Global Relation Extraction, discussed in Chapter 9.

The algorithm for the KG-to-Text module and the Evaluation via the Reverse Engineering module are as follows:

KG-to-Text Algorithm

Input: KG

Output: Text Document

For each triple in the KG,

calculate its relevance score based on the sum of the relevance scores of its subject entity and its object entity.

Sort the triples through their relevance scores.

For each batch of 20 triples in the KG,

get all entities involved in these triples.

[LLaMA Prompt_1] Given a list of (subject, predicate, object) triples in a graph, given the label, aliases, and description of all the entities involved in these triples, prompt the LLM to generate a

11.3 Evaluation via Reverse Engineering

paragraph describing these triples.

Add the paragraph to the output document.

Prompt 1 (Simplified)

Task Definition

You are a linguistic expert involved in the task of knowledge-graph-to-text generation. Given a list of (subject, predicate, object) triples in a graph, given the label, aliases, and description of all the entities involved in these triples, your task is to generate a fluent paragraph describing these triples.

You may use the information of entities for reference. However, the focus of your generated paragraph should be the triples, i.e., the relations between entities, rather than solely the label, aliases and description of each entity.

Output Format

...

Entities

...

Relations (Triples)

...

Output

...

Algorithm for Evaluation via Reverse Engineering

Input: KG (including the input semi-structured document)

Output: Score

Flatten the semi-structured paper stored in the KG into plain text.

[M3] Embed the original paper.

Get the synthesised paper from the KG-to-Text model given the KG.

[M3] Embed the synthesised paper.

11 Evaluation

Score = The cosine similarity between the original paper and the synthesised paper.

The reason we supply the description and aliases of the entities involved in a batch of triples is that the LLM may not know what the entities are since academic papers usually introduce new terms. With the supplement of the entity description, which is part of the output KG of our pipeline, we expect that the LLM can generate text from the triple better. However, we also insert an instruction to prevent the LLM from focusing too much on the description of each entity rather than the triples. This is because the sets of entities involved in each batch of triples are not disjoint. If the synthesised paragraph consists of too much information about the descriptions of each involved entity, the whole synthesised paper will contain a lot of repetitive text.

During the graph-to-text process, no matter which LLM decoder is used to construct the original KGs, the LLM decoder used to generate texts from KGs is always LLaMA (Meta-Llama-3-8B-Instruct-Q4-0), and the LLM encoder used is always BAAI–bge-m3 because of its larger context limit.

Figure 11.10 shows part of the text synthesised from the KG constructed from one of the documents in the Five-Full-Paper dataset for demonstration purposes.

```
data > output > ASKG > kg_test_1_l.json > ...
1  [
2    "iri": "Paper-MEL_Metadata_Extractor__Loader",
3    "title": "MEL: Metadata Extractor & Loader",
4    "authors": [...],
5    "keywords": [...],
6    "sections": [...],
7    "summary": "The metadata and content-based information extraction tasks from heterogeneous file sets are pre-processing steps of kg2text: [
8      "MEL+TNNT tools consist of MEL, a Python-based tool that extracts metadata and content- based information from various file typ
9      "According to MEL, a Python-based tool that extracts metadata and content- based information from various file types, it can as
10     "The set of Python-based methods, which implements MEL are generic and can be applied to extract the content and metadata of al
11     "MEL, a Python-based metadata extractor and loader, uses MEL extracts the textual content from the source document to generate
12     "MEL, as compared to Apache Tika, refers to a lightweight Python-based tool for extracting metadata from various file formats.
13     "MEL, a Python-based tool for metadata extraction and content- based information processing from unstructured file sets, can as
14     "The AGRIF project uses MEL, a Python-based tool that extracts metadata and content-based information from unstructured data se
15     "The process of extracting relevant information from content, known as metadata and content extraction tasks, can be achieved u
16     "The AGRIF project, which uses MEL (Metadata Extractor & Loader) to extract metadata and content-based information from unstruc
17     "The input document sets are categorized into file types, which can be further processed using tools like J2RM and Apache Tika.
18     "The integration of MEL's results with JSON-LD ontologies will be explored in the near future. This involves metadata and conte
19     "The Knowledge Graph Construction Process (KGCP) has a broader term of project, which includes initiatives such as Apache Tika.
20   ],
21   "times": [
22     79.291574716568
23   ]
24 ]
25 ]
```

Figure 11.10: Synthesised Text Example

11.3 Evaluation via Reverse Engineering

Table 11.10: **Similarity Scores** of the text documents synthesised from the KGs constructed from the **Five-Full-Papers** dataset by **LLaMA**

Paper	Length (Token)	Similarity Score
1	1165	0.9395
2	3161	0.8989
3	3062	0.8916
4	1486	0.9224
5	5183	0.8330
Mean/Std	2811 ± 1434	0.8971 ± 0.03626

Table 11.11: **Similarity Scores** of the text documents synthesised from the KGs constructed from the **Five-Full-Papers** dataset by **GPT**

Paper	Length (Token)	Similarity Score
1	1165	0.9155
2	3161	0.9165
3	3062	0.8633
4	1486	0.9302
5	5183	0.8726
Mean/Std	2811 ± 1434	0.8996 ± 0.02655

11.3.2 Evaluation Results

Tables 11.10, 11.11, and 11.12 collectively demonstrate relatively high similarity scores between the original documents and the synthesised documents from the KGs, across all datasets and LLM decoders used. It seems that the similarity score is not quite correlated with the length of the papers in general, although the longest paper (Paper 5) has relatively low similarity scores for both LLM decoders.

The similarity score for documents synthesised from the KGs constructed by GPT is also similar to those by LLaMA.

Although around 0.9 is considered a high similarity score. There is still room for improvement to ensure a more lossless conversion from text documents to KGs.

11.3.3 Remarks

This approach can reflect both the precision and recall of a constructed KG. This is because if the constructed KG does not capture enough information from the original document or the information captured is incorrect, the synthesised document based on the constructed KG cannot be highly similar to the original document.

However, this approach tests both the KGC pipeline and the KG-to-Text model together. It cannot reflect on the performance of our KGC pipeline alone. Since we only

11 Evaluation

Table 11.12: **Similarity Scores** of the text documents synthesised from the KGs constructed from the **SciERC** dataset by **LLaMA** and **GPT**

Paper	Length (Token)	LLaMA	GPT
Mean/Std	134 ± 51	0.8961 ± 0.02930	0.9041 ± 0.04099

implemented a simple KG-to-Text model for our evaluation purpose, the conversion from KGs to texts can also be lossy. The performance of the KGC pipeline can potentially be underestimated.

11.4 Evaluation via Application

The second valuation approach is inspired by the ways how ([Edge et al., 2024](#)) test their constructed KGs. Essentially, this approach comes from one of the largest applications of KGs, Retrieval-Augmented Generation (RAG) ([Pan et al., 2023](#)).

In general, in an RAG system, there is a chatbot, usually implemented by an LLM decoder, that is responsible for answering users' questions. However, instead of directly answering users' questions based on the common knowledge of the chatbot, the RAG system first searches through a knowledge base to find relevant sources. After the retrieval, the relevant sources serving as hints, together with the original user question, are inputted to the chatbot. With the supplied resources, the chatbot can then answer questions more accurately. ([Pan et al., 2023](#))

Traditionally, these sources are in the form of text documents ([Edge et al., 2024](#)). However, KG can also serve as the knowledge base for retrieval. It is expected to be more effective since KGs are structured ([Edge et al., 2024](#)).

Therefore, by evaluating the RAG system that is based on our constructed KGs, we can then indirectly evaluate our KGs and, hence, our KGC pipeline.

However, the drawback is that we have to create an RAG system on top of our KGC pipeline. In fact, graph-based RAG is another field of study that is parallel to KGC in the topic of KG ([Pan et al., 2023](#)). Therefore, creating an advanced Graph-Based RAG system requires additional in-depth study.

Another issue is that neither the Five-Full-Paper dataset nor SciERC have any question-answer pairs to serve as the ground truth.

11.4.1 Evaluation Approach

The solution is to create a simple RAG system for our evaluation. To obtain question-answer pairs, we used one of the most SOTA LLM, gpt-o1-preview, which is considered more advanced but expensive than our currently using Meta-Llama-3-8B-Instruct-Q4-0 and gpt-4o-mini ([OpenAI, 2024b](#)). Specifically, we first let the GPT-O1 read the

11.4 Evaluation via Application

whole document for all papers in the Five-Full-Paper dataset. This is practical because GPT-O1 has a wide input limit of 128K tokens. Then, we ask GPT-O1 to generate 10 question-answer pairs. These question-answer pairs are then served as the inputs and targeted outputs for the evaluation of our RAG system.

These question-answer pairs are expected to be highly accurate because they are generated by the LLM through reading the whole original document.

On the RAG side, the system can only use the constructed KG to retrieve relevant facts for the chatbot to answer questions. We also limit the number of hops the system can query the KG from an entity node relevant to the question. This is because it is somewhat impractical to use the whole KG as supplementary resources which would undermine the RAG approach of retrieving only the most relevant facts.

Since papers in SciERC are too short, this evaluation approach only underke on the Five-Full-Paper dataset.

The prompt used for creating the question-answer pairs is as follows:

Prompt for Generating Question-Answer Pairs (Simplified)

Task

You are a university teacher who has recently assigned your students to read an academic paper. To test their understanding, you will prepare 10 questions that are only answerable by referring to the paper. Your questions should be objective, meaning each question should potentially have a clear, concrete answer without the need for subjective interpretation. You should also provide a sample answer for each question.

Output Format

...

Paper

...

Your Response

...

We created the scenario that a university teacher is creating questions to test their student about the understanding of the paper because reading papers is a common activity for both teachers and students in universities.

We included the instruction that “questions should be only answerable by referring to the paper” to avoid our chatbot simply using its common knowledge to answer the question. If such a case happens, we then cannot effectively evaluate our KGs.

11 Evaluation

We included the instruction that “questions should be objective” to prevent the LLM from generating questions that require the opinions of the chatbot instead of solely based on the content of the paper.

Next, we developed a simple RAG system that is capable of answering questions regarding the papers.

Specifically, we have implemented both the text-based RAG and graph-based RAG for head-to-head comparison. Their algorithms are as follows:

Text-Based RAG

Input: A question regarding the paper
A semi-Structured Paper
(This is the same as the input of our pipeline)

Output: An answer

[BGE] Embed all sentences in the input paper.

[BGE] Embed the question.

Retrieve the 10 most relevant sentences from the input paper based on the similarities between the embeddings of sentences and the question.

[LLaMA Prompt_1] Given the question and 10 most relevant sentences in the paper, ask the LLM decoder to generate an answer.

Graph-Based RAG

Input: A question
The KG of a paper
(This is the same as the output of our pipeline)

Output: An answer

[M3] Embed all entities in the KG based on their labels.

[LLaMA Prompt_2] Extract all keywords from the questions.

Embed all keywords.

For each keyword,
find the closest entity in the KG.
Store in relevant_entities.

For each relevant entity in relevant_entities,
find all one-hop triples that involve this entity.

```
Store in relevant_triples.

For each relevant triple in relevant_triples,
    if the subject or the object involved is not in relevant_entity,
        add it into relevant_entity.

[LLaMA Prompt_3] Given a list of relevant triples, the label, aliases,
and description of the relevant entities, and the original question, ask
the LLM to answer it.
```

Prompt 3 (Simplified)

```
## Task
```

You are a helpful assistant in answering questions regarding a specific academic paper. Given a question about an academic paper and some resources extracted from the knowledge graph of the paper, your task is to try to answer the question. The resources you are given are a list of relevant entities and a list of (subject, predicate, object) triples about their relations.

```
## Output Format
```

...

```
## Question
```

...

```
## Resources (Entities)
```

...

```
## Resources (Triples)
```

...

```
## Answer
```

...

We also supply the description of entities when supplying triples for a similar reason to the prompt used for KG-to-Text in Section 11.3. The entity's label alone can be inefficient and potentially ambiguous.

The text-based RAG here is highly important because it essentially tests how the current stage of ASKG, without the deep structured knowledge about the content of the paper,

11 Evaluation

can be used to perform RAG. If RAG is the main application of ASKG, this may lead to a discussion about whether it is necessary to put so much effort into creating KGs or we may maintain the current stage of ASKC, i.e. in the form of a metadata graph instead of KG of the content of papers.

Last, we check the similarity between the generated answers from our RAG system with the ground truth answers.

Overall, the diagram for our Evaluation via Application is shown in Figure 11.11. This diagram essentially shows the general picture of one of the potential applications of KGs.

Note that we have used GPT-O1-Preview to directly read and create questions and answers for a document. This is only for evaluation purposes to generate question-answer pairs and is not quite a practical approach to querying a document in real life. This is because a RAG system in real life usually searches through multiple documents instead of just one. In this case, it can be impractical to feed all documents into the LLMs to directly generate answers. Instead, we should rely on KGs or Metadata Graphs to effectively retrieve only the key information. This is why creating KGs is important even though the current state of LLMs can directly answer the question by looking at the whole single paper.

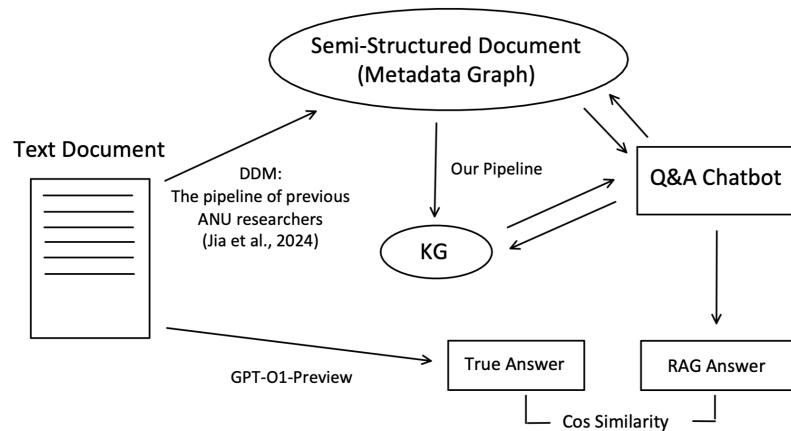


Figure 11.11: The process of evaluation via application

Figure 11.12 shows a sample list of tuples of question, target answer, RAG answer, and their similarity scores for demonstration purposes.

11.4.2 Evaluation Results

Figure 11.13 and Table 11.13 show the comparison of the similarity score between the target answers and the answer generated by the RAG system using the three different knowledge bases.

11.4 Evaluation via Application

```

1 [ {
2   "question": "What is the main purpose of the MEL tool introduced in the paper?",
3   "answer": "The main purpose of MEL is to implement a set of methods to extract metadata and content-based information from unstructured documents",
4   "LLM_answer": "The main purpose of the MEL tool introduced in the paper is to extract metadata and content-based information from unstructured documents",
5   "similarity_score": 0.93798828125
6 },
7   {
8     "question": "How does MEL integrate with TNNT, and what is the benefit of this integration?",
9     "answer": "MEL is integrated with The NLP-NER Toolkit (TNNT) to automate the extraction of categorized named entities from the input text",
10    "LLM_answer": "MEL integrates with TNNT by combining MEL's capabilities for extracting metadata and content-based information from unstructured documents with TNNT's named entity recognition capabilities",
11    "similarity_score": 0.90478515625
12 },
13   {
14     "question": "What are the main differences between MEL and Apache Tika?",
15     "answer": "While both MEL and Apache Tika aim to extract metadata and content from various file formats, the main differences are in their design and implementation",
16     "LLM_answer": "The main differences between MEL and Apache Tika are as follows: MEL is a Python-based tool specifically designed for structured document processing, while Apache Tika is a Java-based tool for unstructured document processing",
17     "similarity_score": 0.9609375
18 },
19   {
20     "question": "How many different file types does MEL support, and what is the significance of this feature?",
21     "answer": "MEL supports more than 20 different file types. This comprehensive support makes it a versatile tool capable of processing various types of documents",
22     "LLM_answer": "MEL supports more than 20 different file types. This feature is significant because it allows the tool to extract metadata from a wide range of document formats",
23     "similarity_score": 0.95849609375
24 },
25   {
26     "question": "What are the four major contributions of MEL as highlighted in the paper?",
27     "answer": "The four major contributions of MEL are: (1) the ability to extract metadata sets and content-based information from unstructured documents, (2) the use of a modular architecture, (3) the implementation of advanced natural language processing techniques, and (4) the provision of a user-friendly interface",
28     "LLM_answer": "The four major contributions of MEL highlighted in the paper are: 1) Comprehensive metadata extraction support for various file formats, 2) Modular design for easy customization, 3) Integration with other NLP tools like TNNT, and 4) User-friendly command-line interface",
29     "similarity_score": 0.9462890625
30 },
31 ]

```

Figure 11.12: A list of tuples of question, target answer, RAG answer, and similarity score

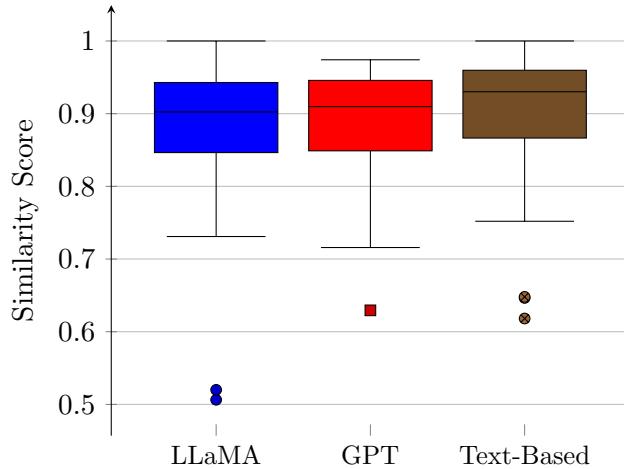


Figure 11.13: Comparison of the **Similarity Scores** between the target answers and the answer generated by the RAG system using the KG constructed by LLaMA (Graph-Base RAG), the KG constructed by GPT (Graph-Based RAG), and the original semi-structured document (Text-Base RAG)

The score of the RAG system based on the original semi-structured paper seems to be higher although it is not statistically higher.

11 Evaluation

Table 11.13: Comparison of the **Similarity Scores** between the target answers and the answer generated by the RAG system using different knowledge bases

RAG Method	Using KG by LLaMA	Using KG by GPT	Using the Original Semi-Structured Paper
Mean/Std	0.8842 ± 0.09523	0.8884 ± 0.07622	0.9018 ± 0.08873

However, with limited testing, we cannot say that one is better than others because there are many factors.

- RAG in real life is usually based on multiple documents; the text-based RAG may not scale well compared to Graph-Based RAG because, with proper Knowledge Graph Alignment, the same entities of different local KGs of papers should be linked to each other. Therefore, the keywords of the question can actually match the entities and start from the entities, many facts about the entities coming from different documents can be retrieved. In traditional RAG, there may not be such as global view of an entity.
- Both the Graph-Base RAG and Text-Base RAG are considered the baseline approach. Thus, they do not reflect on the true potentiality of Graph-Base RAG and Text-Base RAG.

Even though, it is still worthy to have such side by side comparsion.

Nevertheless, the test results at least show that the KG constructed by our pipeline is useful in RAG as it can be used as the knowledge base to help the chatbot generate answers highly similar to the target answer, reflecting on a potentially high quality of the constructed KG and hence the performance of the KGC pipeline.

11.4.3 Remarks

This evaluation approach should also reflect both the *precision* and the *recall* of the KG. This is because the questions are directly relevant to the documents. To allow the answer to be more correct, the KG should precisely and thoroughly capture the information of the original document.

However, similar to the Evaluation via Reverse Engineering approach in Section 11.3. This approach cannot test the KGC pipeline alone. Both the KGC pipeline and RAG system can affect the overall performance.

This evaluation approach also tests the *conciseness* of the KG, i.e. the performance of the Stage: Coreference Resolution and Entity Disambiguation (7), or how well the KGC pipeline can merge two mentions referring to the same thing into a single entity.

In our testing, we deliberately restricted our RAG system to retrieve only one closest

entity for each keyword. For example, if “ANU” is a keyword in the question, only the closest entity, e.g., “ANU”, will be returned, and the triple returned is further based on the matched entities only. Therefore, if “The Australian National University” does not merge with “ANU” as a single entity, the information (triples) about “The Australian National University” will be not be retrieved.

Therefore, without proper Coreference Resolution, the resultant KGs may still achieve a high score in the Evaluation via Reverse Engineering but not in the current Evaluation via Application.

11.5 Manual Evaluation

The manual evaluation is our tertiary evaluation approach because it is subjective and requires human effort, which is not easy to undertake and potentially lacks reliability. However, it can serve as the use case to demonstrate graphically how the actual KGs constructed by our KGC pipeline look compared to KGs constructed by the traditional KGC approach.

Since the KGs for a full paper can be significantly large. The manual evaluation is performed on SciERC, the abstract-only paper dataset. Specifically, we selected *the best* paper and the *worst paper* during the Evaluation via Reverse Engineering 11.3 to have a close look at these KGs. “The best” here means the highest similarity score between the synthesised document from the KG and the original document.

The best KG constructed by the pipeline using LLaMA in SciERC is Paper NO.83. It achieved a similarity score of 0.98 in Evaluation via Reverse Engineering. It is shown in Figure 11.14

The worst KG constructed by the pipeline using LLaMA in SciERC is Paper NO.64. It achieved a similarity score of 0.77 in Evaluation via Reverse Engineering. It is shown in Figure 11.16

Their target KGs provided by SciERC is shown in Figure 11.15 and Figure 11.17, respectively.

The predefined entity types of the target KG are Method, Material, Task, Metric, Generic, and OtherScientificTerm.

The predefined predicates of the target KG are FEATURE-OF, PART-OF, CONJUNCTION, COMPARE, USED-FOR, EVALUATE-FOR, and HYPONYM-OF.

Their original texts are as follows (Luan et al., 2018):

Paper NO.83

In this paper we present our recent work on harvesting English-Chinese bitexts of the laws of Hong Kong from the Web and aligning them to the

11 Evaluation

subparagraph level via utilizing the numbering system in the legal text hierarchy.

Basic methodology and practical techniques are reported in detail.

The resultant bilingual corpus, 10.4 M English words and 18.3 M Chinese characters, is an authoritative and comprehensive text collection covering the specific and special domain of HK laws.

It is particularly valuable to empirical MT research.

This piece of work has also laid a foundation for exploring and harvesting English-Chinese bitexts in a larger volume from the Web .

Paper NO.64

Techniques for automatically training modules of a natural language generator have recently been proposed, but a fundamental concern is whether the quality of utterances produced with trainable components can compete with hand-crafted template-based or rule-based approaches.

In this paper, We experimentally evaluate a trainable sentence planner for a spoken dialogue system by eliciting subjective human judgments.

In order to perform an exhaustive comparison, we also evaluate a hand-crafted template-based generation component, two rule-based sentence planners, and two baseline sentence planners.

We show that the trainable sentence planner performs better than the rule-based systems and the baselines, and as well as the hand-crafted system.

For Paper NO.83, the KGs constructed by the pipeline consist of more entities and relations than the target KG provided by SciERC. Since the target KG only captures entities that fall within a predefined type, it does not capture entities like “this paper” and “Hong Kong”. The target KG also does not capture relations like (“The resultant bilingual corpus”, “is”, “an authoritative and comprehensive text collection”).

Both graphs capture the key information that the bilingual corpus is used for empirical MT research. The target KG contains the triple (“bilingual corpus”, ”used for”, “empirical MT research”), whereas the constructed KG contains the triples (“It”, “is valuable to”, “empirical MT research”).

The relations in the target KG are more general and concise, while the relations in the constructed KG are more specific to the text.

One key observation is that in constructed KG, the entity “It” does not successfully merge with the entity “bilingual corpus,” which reduces the conciseness of the graph. As

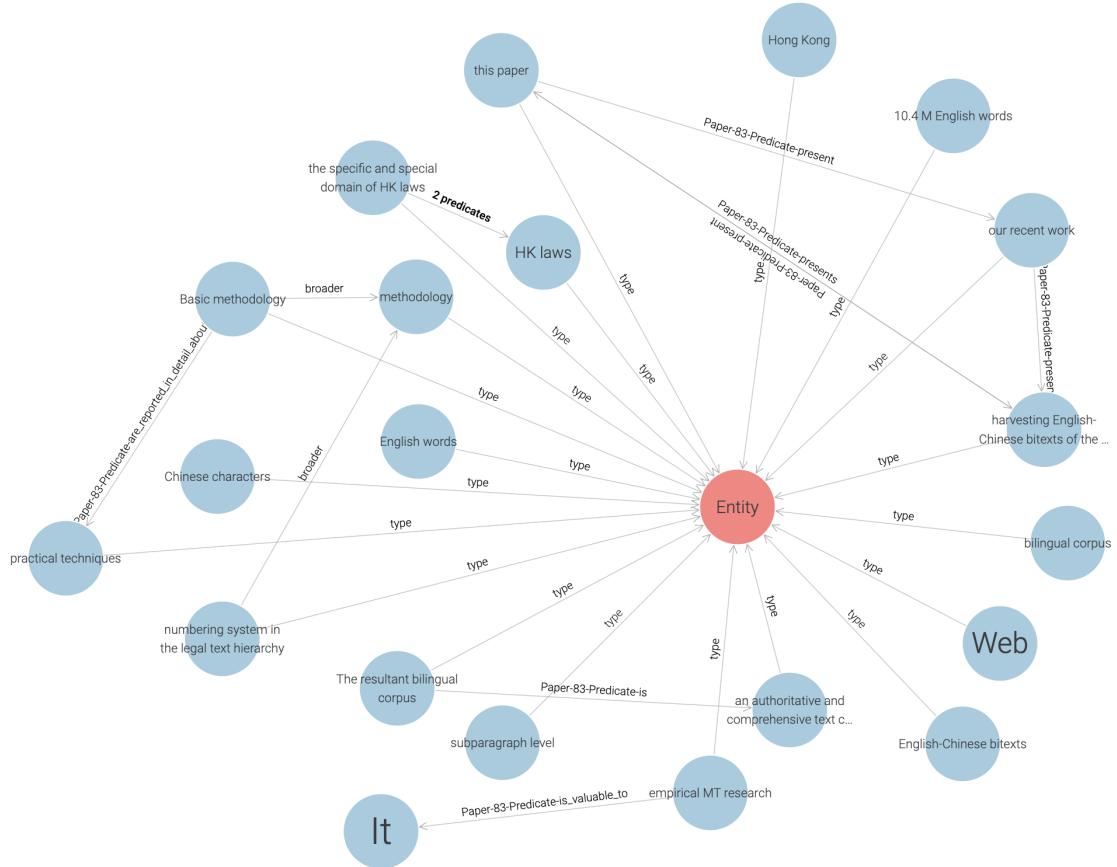


Figure 11.14: Constructed KG for Paper No.83

11 Evaluation

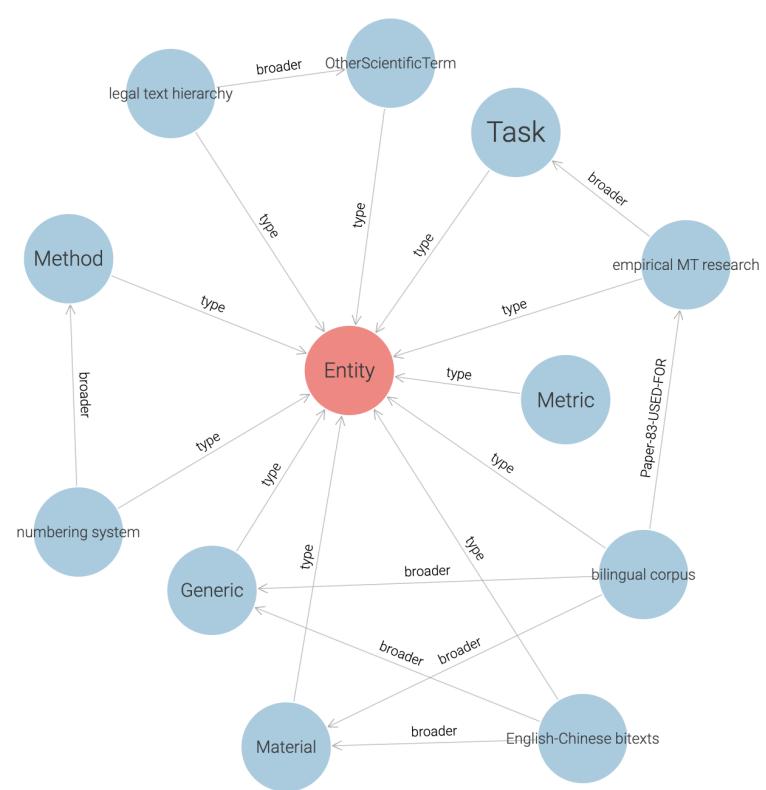


Figure 11.15: Target KG for Paper No.83

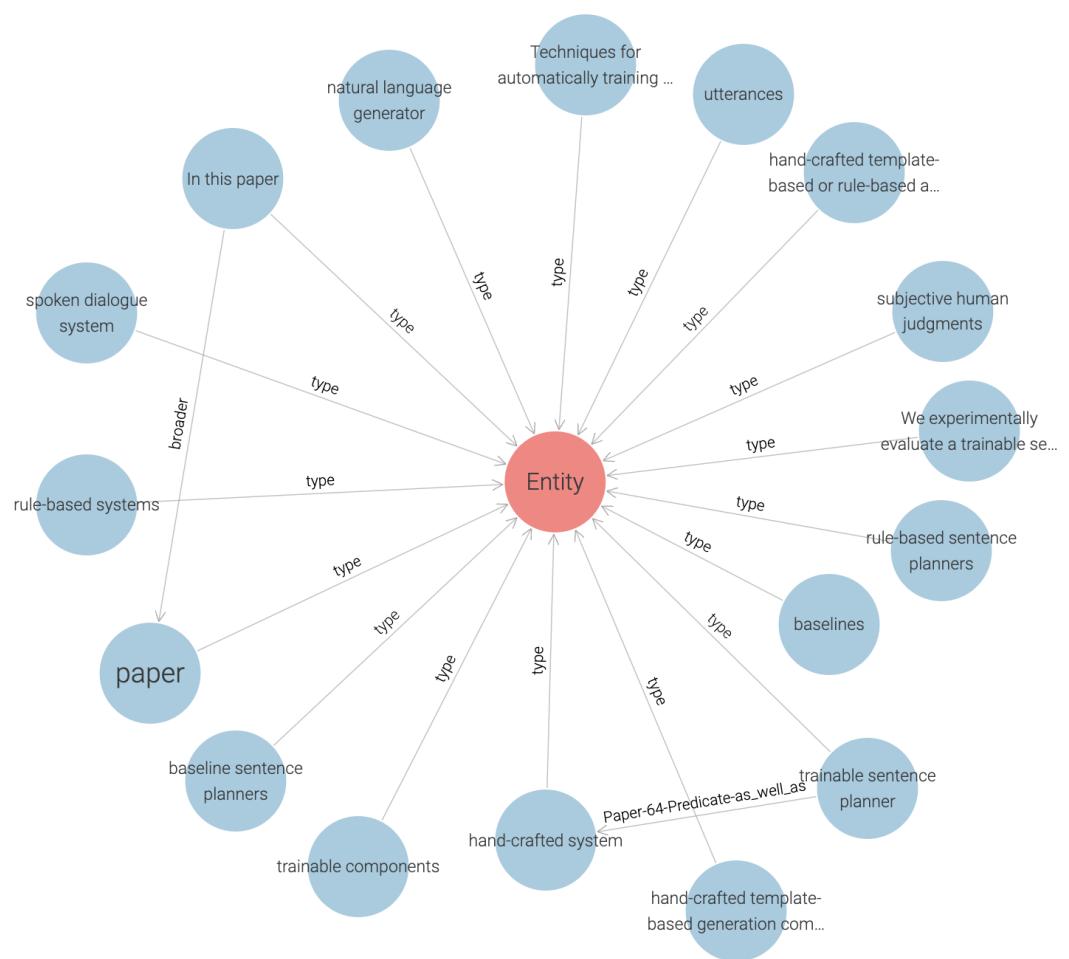


Figure 11.16: Constructed KG from Paper No. 64

11 Evaluation

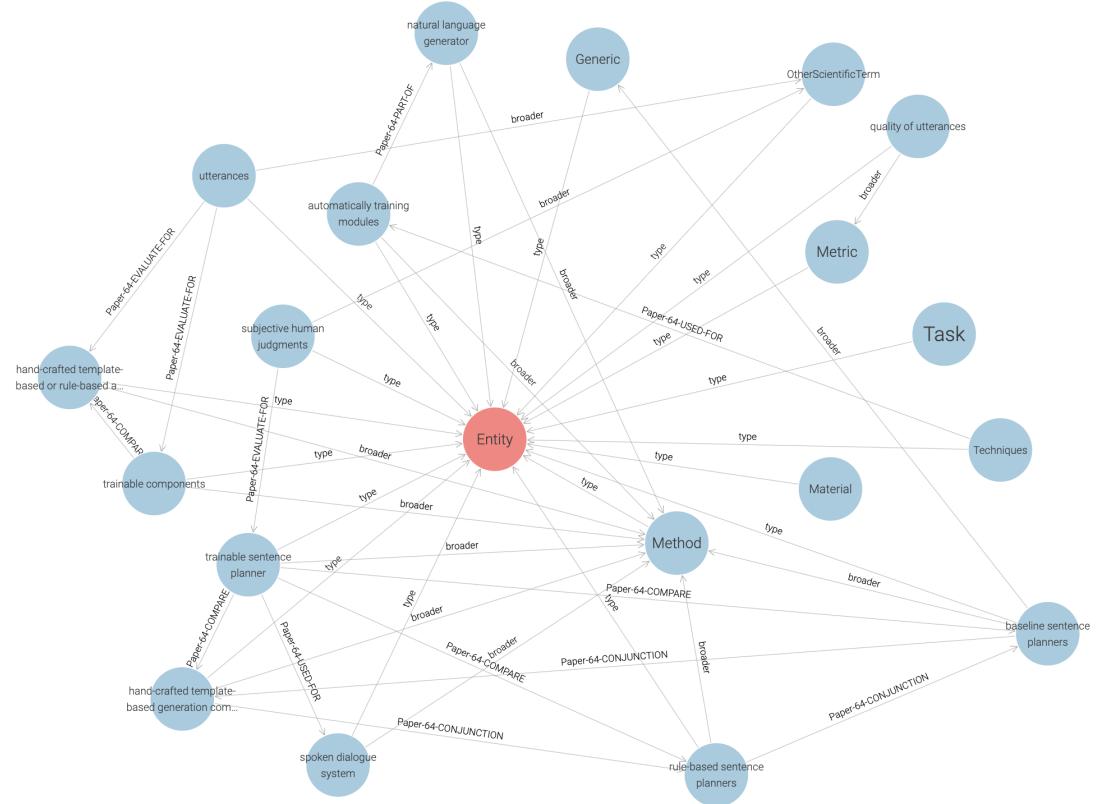


Figure 11.17: Target KG from Paper No. 64

Table 11.14: **Statistics** of the KGs constructed from the **Five-Full-Papers** dataset by **LLaMA**: Original and Ablation Study

Paper	Length (Token)	Entities	Mentions	Relations	Isolated Entities
Original	2811	356	682	789	74
	±	±	±	±	±
	1434	184	384	404	31
Ablation Study	2811	332	682	662	75
	±	±	±	±	±
	1434	166	384	342	32

mentioned in the last section (11.4), Evaluation via Reverse Engineering fails to evaluate the performance of the stage of Coreference Resolution. Even though this constructed KG achieves a high similarity score when reverted back to text, it does not mean the KG is concise.

For Paper NO.64, shown in Figure 11.16 and Figure 11.17, the target KG and the constructed KG contain a similar set of entities. However, the constructed KG fails to contain many relations the original paper has and includes in the target KG.

This is mainly due to the algorithm of Stage 3: Local Relation Extraction of the pipeline (3). We have forced all triples extracted by the LLM to appear explicitly in the document, both subject and object. The LLM somehow does not follow it, leading to many triples extracted being filtered out. However, if we did not filter out the triple with both its subject and object explicitly appearing in the original document, there is also a possibility that the triple is just hallucinated by the LLM and does not actually consist in the original document. Therefore, this is a trade-off.

11.6 Ablation Study

The general evaluation section (11.2.1) has suggested the removal of the LLM double-checking mechanism during the Coreference Resolution stage to aim to reduce the complexity from $O(2M + M^2)$ to $O(2M)$. Therefore, this section aims to see whether the LLM double-checking mechanism is significant to the pipeline by repeating the Evaluation via Reverse Engineering (11.3) and Evaluation via Application (11.4) on the KGs constructed by the pipeline with the removal of the LLM double-checking component.

Specifically, we only focus on the KGs constructed from the Five-Full-Paper dataset by the pipeline using LLaMA.

Tables 11.14 and 11.15 show the general statistics about the KGs constructed by the pipeline without the LLM double-checking mechanism and the KGs constructed by the original pipeline. It was found that the reduced pipeline has the same mentions as the original pipeline but fewer entities for all papers in the dataset. This is because the reduced pipeline merges multiple mentions into the same entity in the Coreference Stage

11 Evaluation

Table 11.15: The number of **Relations** of the KGs constructed from the **Five-Full-Papers** dataset by the **Original Pipeline** and **Reduced Pipeline** based on **LLaMA**

Paper	Relations (Original Pipeline)	Relations (Reduced Pipeline)
1	235	212
2	860	700
3	894	789
4	518	404
5	1437	1207
Mean/Std	789 ± 404	789 ± 404

Table 11.16: **Runtimes** (in Minutes) of KGC from the **Five-Full-Papers** dataset by **LLaMA**: Original and Ablation Study

Paper	S0	S1	S2	S3	S4	S5	S6	Time
Original	3.320E-6	16.66	122.9	9.074	28.51	11.06	3.729E-3	188.2
	±	±	±	±	±	±	±	±
	1.228E-6	8.574	120.5	4.665	9.551	4.256	1.604E-3	142.6
Ablation Study	3.320E-6	16.66	30.59	9.173	27.49	11.00	4.531E-3	94.92
	±	±	±	±	±	±	±	±
	1.228E-6	8.574	19.84	4.702	10.01	4.325	2.239E-3	45.31

more aggressively. However, it also led to fewer relations extracted. This is potentially due to two potential reasons.

1. One is that multiple mentions referring to different things are incorrectly merged into the same entity. Since only one description is kept during the merging, the description itself may not be correct. During both global and local relation extraction, descriptions are the key recourse to help the LLM understand an entity in order to extract the relation between this entity and other entities in a text. Since the description may not correctly describe the entity, LLM may be confused about what the entity is, and hence, fewer relations are extracted from this wrong entity.
2. The second reason is that the original LLM double-checking mechanism is too conservative, leading to some mentions referring to the same entity being treated as different entities. Therefore, some relations extracted can be redundant, e.g., (“ANU”, “is a”, “University”) and (“The Australian National University”, “is a”, “University”) if the mentions “ANU” and “The Australian National University” are not treated as the same entity. Therefore, there is a trade-off.

As shown in Tables 11.17 and 11.16, the runtime of Stage 2, Coreference Resolution and Entity Disambiguation, has been reduced significantly for all papers with the removal of the double-checking mechanism.

11.6 Ablation Study

Table 11.17: **Runtimes** (in Minutes) of KGC from the **Five-Full-Papers** dataset by the **Original Pipeline** based on **LLaMA**

Paper	Paper Length	S2 (Original Pipeline)	S3 (Reduced Pipeline)
1	1165	20.18	8.374
2	3161	117.7	34.31
3	3062	79.03	34.13
4	1486	42.93	12.25
5	5183	354.6	63.92
Mean/Std	2811 ± 1434	122.9 ± 120.5	30.59 ± 19.84

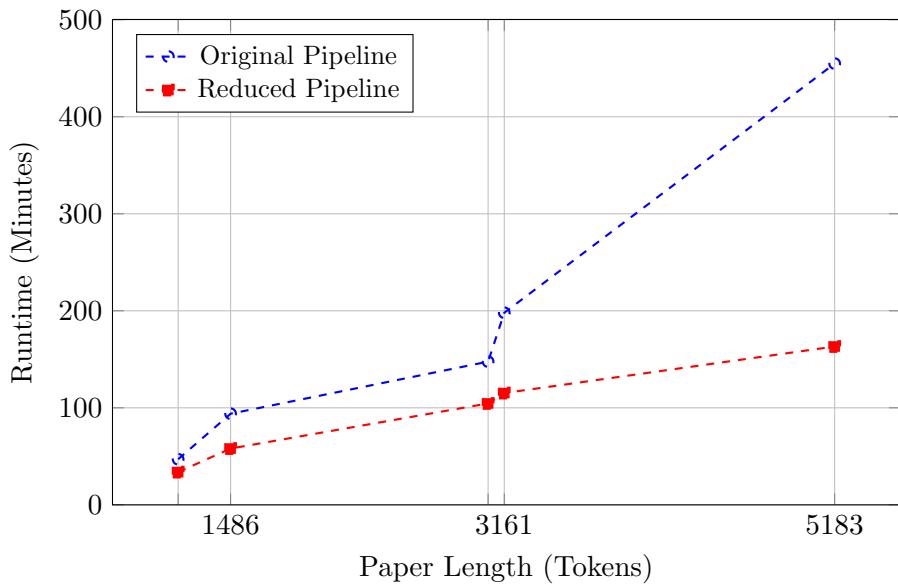


Figure 11.18: Runtime Comparison of Original and Reduced Pipelines vs. Paper Length

Figure 11.18 shows the total runtime of the pipeline with respect to papers of different lengths. It was found the runtime became approximately linearly proportional to the paper length. This aligns with the estimated pipeline complexity. By reducing $O(2M + M^2)$ to $O(2M)$, all *quadratic* components in the pipeline are removed. The complexity of the rest of the stages is all linearly proportional to the paper length, as discussed in the General Evaluation section (11.2.1). Even so, the time used to construct KG from papers is quite long, e.g., 2.72 hours for a 5183-tokens paper. Therefore, this is due to the fact that the pipeline is mainly based on the LLM decoder, and the LLM decoder itself has high complexity, leading to a high proportionality constant of the pipeline complexity. Therefore, further reducing complexity can be part of future work, e.g., reducing the constant proportionality or targeting a sub-linear relationship.

11 Evaluation

Table 11.18: Comparison of the **Similarity Scores** between the **original papers** and the **papers synthesised from the KGs** constructed by the **original pipeline** and the **reduced pipeline** based on **LLaMA**.

Paper	Original Pipeline	Reduced Pipeline
1	0.93945	0.9272
2	0.89893	0.8515
3	0.8916	0.8477
4	0.9224	0.9185
5	0.8330	0.7573
Mean/Std	0.8971 ± 0.03626	0.86044 ± 0.06116

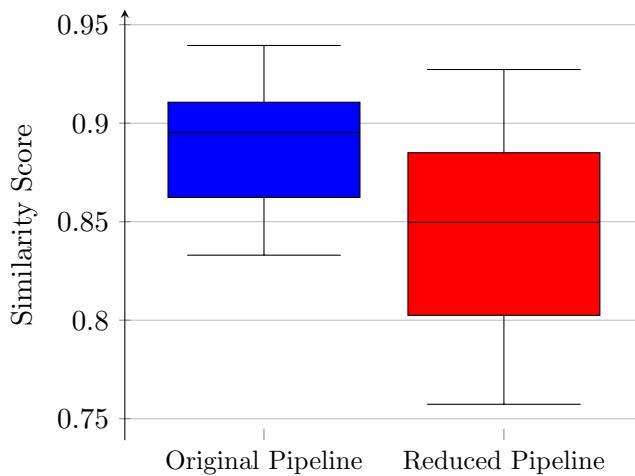


Figure 11.19: Boxplot of the **Similarity Scores** between the **original papers** and the **papers synthesised from the KGs** constructed by the **original pipeline** and the **reduced pipeline** based on **LLaMA**.

Table 11.18 and Figure 11.19 show the similarity scores between the original papers and the papers synthesised from the KGs constructed by the original pipeline and the reduced pipeline. Both the original pipeline and the reduced pipeline are based on LLaMA. Both KGs are constructed from the Five-Full-Paper dataset. It was found that the scores slightly decreased for all papers for the reduced pipeline. From the boxplot, it was found that the overall score is lower, and the standard deviation is higher. However, given the limited samples, we cannot claim that one is statistically different from the other because their p-value (0.333) is not small enough.

Therefore, this further shows that removing the double-checking mechanism may not significantly affect the performance estimated through the Evaluation via Reverse Engineering approach.

Table 11.19 and Figure 11.20 show the similarity scores between the target answers

Table 11.19: Comparison of the overall **Similarity Scores** between the target answers and the answer generated by the RAG system using the KGs generated by the **original pipeline** and the **reduced pipeline** based on **LLaMA**.

	Original Pipeline	Reduced Pipeline
Mean/Std	0.8842 ± 0.09523	0.8704 ± 0.09832

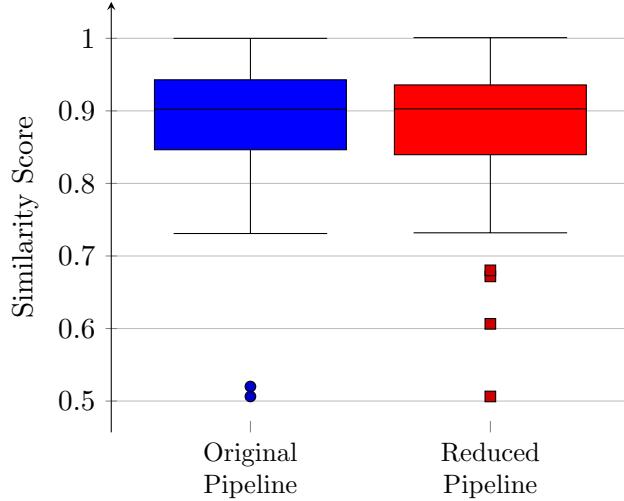


Figure 11.20: Boxplot of the **Similarity Scores** between the target answers and the answer generated by the RAG system using the KGs generated by the **original pipeline** and the **reduced pipeline** based on **LLaMA**.

and the answer generated by the RAG system using the KGs generated by the original pipeline and the reduced pipeline. Both pipelines are based on LLaMA, and both KGs are constructed from the Five-Full-Paper dataset. It was found that the overall results are quite similar. However, as shown in the boxplot, for the reduced pipeline side, there are more outliers. An outlier, especially for those with low similarity, potentially means the chatbot almost fails to answer the question, which is further due to the incorrectness or incompleteness of the KGs. Therefore, removing the double-checking mechanism may slightly reduce the quality of the RAG and thus further affect the performance of the RAG system. However, the effect is not statistically significant, given the limited samples.

Overall, the ablation study shows that removing the double-checking component in Stage 2, Coreference Resolution and Entity Disambiguation, does not significantly affect the performance of the pipeline based on the current dataset. However, a slight decrease in performance has been detected for both evaluation methods.

11.7 Summary of Limitations

The major limitation during the evaluation process is that the datasets used are too small, both for the sizes and numbers of samples. Only five full papers are used. The rest of the 100 papers are abstract only.

- This is mainly due to the fact that the current complexity of the pipeline is still high, and it can take a lot of time to build even a single local KG for a paper.
- The second reason is that by the start of our evaluation process, we were only given ten semi-structured papers from the ASKG. Since our pipeline does not start from raw text documents, it is hard for us to have more samples. Within these ten documents, one is used as a validation set, and 4 of them are preprocessed and used in our evaluation. The remaining six are still under preprocessing as we have found some major issues in these semi-structured documents, such as the incorrectness of indices of sentences and paragraphs, which can significantly affect later KGC.

The second limitation is the embeddings of documents themselves can be lossy. Both Evaluation via Reverse Engineering and Evaluation via Application heavily rely on the similarity score between the embedding of the predicated results and the ground truth results. However, even though their embeddings are found similar, the predicated results and the ground truth result may not be completely similar. Therefore, the results are only the estimation of their similarity.

The third limitation is that both Evaluation via Reverse Engineering and Evaluation via Application evaluate not only the KGC pipeline but also other components together. Therefore, the results may not clearly show the actual performance of the pipeline.

The fourth limitation is the narrow author group and topic of these papers. In the Five-Full-Paper dataset, all papers are in the field of computer science. Additionally, four of them are, in fact, the papers written by the supervisors of this project. Given the limited author group and topic, the linguistic patterns of these documents can also be limited. Different linguistic patterns can lead to different performances. Some may even cause the pipeline to fail if they are unhandled edge cases. Therefore, the evaluation may not truly affect the performance of the KGC pipeline of a variety of paper documents.

Chapter 12

Concluding Remarks

This chapter will provide a final overview of the study, summarising the research background, KGC pipeline design, implementation, and evaluation discussed thus far. It will also provide direction for future work and highlight the ethical considerations of this study.

12.1 Research Summary

This study starts with the motivation that the current state of the ASKG lacks a deep semantic representation for the papers it stores, i.e., the KG representations of the paper contents. However, directly creating a single global KG representing the contents of all papers in ASKG may not be feasible. Therefore, this study focuses on finding a way to create local KGs representing a single semi-structured paper in ASKG. In contrast, the task of aggregating local KGs into a more global KG, i.e., the Knowledge Graph Alignment, has been left as future work.

However, the conversion from a single full paper document into its local KG representation can still be challenging. The traditional automatic local KGC approach from text documents focuses on extracted named entities and their relations, which fall into a pre-defined set of types. However, this may not be enough for building KG from academic papers because academic papers tend to introduce new things and knowledge that is potentially outside of the predefined types of named entities and relations. In addition, not only the relations between two named entities but also the relations between concepts and entities and between two concepts are important. Therefore, we process a holistic local KGC approach that aims to capture more knowledge from the original document, especially academic papers.

To turn the holistic local KGC approach into an actual pipeline. We used both LLM encoders and decoders, but mainly LLM decoders. In each stage of the KGC approach,

12 Concluding Remarks

some pre-written prompts are sent to an LLM decoder to direct it to achieve the NLP subtasks required in KGC, e.g., Entity Recognition and Coreference Resolution. Such a way is referred to as In-Context Learning or Prompt Engineering, which removes the need for training and makes our pipeline more independent of the specific LLM used. However, careful design of prompts becomes essential to allow LLM to work as expected.

After the local KGs are constructed by the pipeline, the evaluation of these KGs, and hence, the pipeline, comes into two main approaches. One is *Evaluation via Reverse Engineering* and the other is *Evaluation via Application*. *Evaluation via Reverse Engineering* evaluates the invertibility of the constructed KGs by the pipeline back to the original text document, which indirectly reflects the precision and recall of these KGs. *Evaluation via Application* evaluates how well the constructed KGs can help in an RAG system, which also indirectly evaluates the quality of the KGs and, hence, the performance of the pipeline. The pipeline performance is also evaluated by its runtime. It was found that except for Stage 2, Coreference Resolution and Entity Disambiguation, which scale quadratically with the paper length, all the runtimes of other stages are approximately linearly proportional to the paper length. Therefore, an ablation study is performed to evaluate the pipeline with the removal of the double-checking component in Stage 2. It was found that there is no significant decrease in performance, but both the overall complexity and measured runtime of the pipeline have been reduced to linear. However, the evaluation is done on limited datasets, and both evaluation methods do not evaluate the KGC pipeline alone but with other components, e.g., the RAG system and the Graph-to-Text model together, which may not reflect the true performance of the KGC pipeline.

12.2 Conclusion

As mentioned in the Introduction chapter (Chapter1), this study comes with two main research goals listed below:

1. Design and implement a local holistic KGC pipeline prototype that takes semi-structured academic papers as inputs.
2. Briefly discover the effectiveness of using LLMs in local holistic KGC through the evaluation of the output KGs of the pipeline prototype.

Clearly, a functional prototype of the local holistic KGC pipeline has been successfully implemented. However, there is still room for improvement.

1. Many options we come up with to implement the pipeline are mainly based on some heuristics and assumptions drawn from common knowledge and trial-and-error rather than literature, such as the detailed way of writing prompts for each NLP task in each stage. This is because most of the literature we reviewed focuses more on the high-level approach of KGC. Therefore, these options themselves may lack validation.

12.3 Future Work

2. Even with a number of candidate options to construct the pipeline, the formal validation procedure may still require, e.g., random search and grid search to test different combinations of these options. However, in this study, our search space is quite limited due to the time constraint. This can cause our pipeline to converge to some sub-optimal state, just like the insufficiency of hyperparameter tuning of models in traditional machine learning.
3. Since KGC involves various NLP tasks, it intersects significantly with linguistics. However, due to limited background knowledge and insufficient literature review in linguistic aspects, our pipeline may also converge to a suboptimal state.

As for the effectiveness of using LLMs, especially LLM decoders, in local holistic KGC, based on the evaluation results, it was found that LLM decoders can indeed be used in the construction, and the results based on two main evaluation approaches are acceptable. However, the major issue is the complexity of each LLM decoder call, which is more than quadratical as modelled in the background section (Section 2). Even though the complexity of our pipeline has been reduced after the ablation study, it has been reduced to linear. The proportionality constant is still quite high due to the nature of LLM decoders. Therefore, there is still a gap in making a KGC system that is mostly based on LLM decoders into practice, compared to traditional KGC systems that are mostly based on fine-tuned LLM encoders or other more efficient tools.

12.3 Future Work

There are two major directions for future work extending this research. One is “scaling up”, and the other is “scaling out”. “Scaling up” means improving the performance of the KGC pipeline itself. “Scaling out” is to create a complete ecosystem for the ASKG, from the upstream tasks like constructing and aligning local KGs from more documents to enrich the ASKG to downstream tasks like enabling more applications of ASKG, e.g., Graph-Base RAG.

12.3.1 “Scaling Up”

As suggested in the Conclusion section, there are several ways to develop a more advanced LLM-based holistic local KGC pipeline in the future.

1. One way is to develop more sophisticated experiments to validate each of the current components of the pipeline, as well as different candidate options. This can ensure the pipeline converges to some optimal solutions in terms of the KG quality and runtime performance.
2. Another way is to have people with strong linguistic backgrounds involved in the research and development of the pipeline. This collaboration may introduce additional implementation options, incorporating domain knowledge from both computer science and linguistics.

12 Concluding Remarks

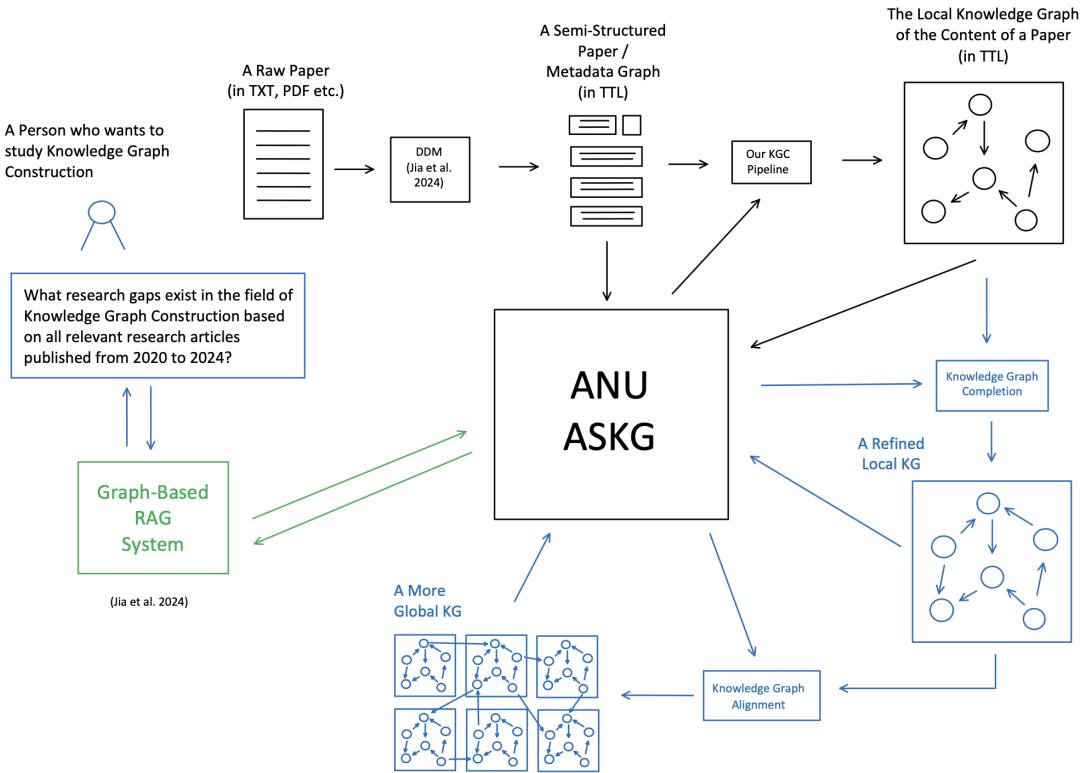


Figure 12.1: An example ecosystem for the ASKG

12.3.2 “Scaling Out”

Figure 12.1 shows how an ecosystem for the ASKG potentially looks like, where the components marked as blue are still under research, and the components marked as green are partially implemented. One of the potential applications of ASKG is the Graph-Base RAG system. There are many questions in real life, especially questions about cutting-edge technology that cannot be answered thoroughly and accurately solely based on the limited or potentially outdated background knowledge of an LLM chatbot like ChatGPT. Therefore, we need KGs that store every single piece of structural facts for an LLM chatbot to reference before answering the question. However, to enable the advanced graph-based RAG system, a sophisticated knowledge base is required.

In this study, we only research constructing coal KG from single papers. We have also briefly touched on the graph-based RAG system during the evaluation, but the RAG system developed is rather basic and can only query a single local KG. After a local KG is constructed, there are still some steps before putting them into use. In fact, each step is considered another deep topic within the field of KG (Pan et al., 2023) and requires significant study. For example, Knowledge Graph Completion aims to refine the constructed KG. The refinement includes but is not limited to predicting the missing

link between entities based on knowledge inferred from a KG and fixing any incorrect or contradicted triples ([Hogan et al., 2020](#)). Knowledge Graph Alignment aims to align multiple KGs into a larger knowledge base ([Pan et al., 2023](#)). For example, this step may involve linking the entity “ANU” among different KG. In this case, when someone wants to search for the entity “ANU”, they can be directed to all local KGs consisting of the entity “ANU”, and hence retrieve the information of “ANU” from more than a single source. Since different documents may have different judgments about the same entity, querying multiple aligned local KGs is more likely to find a neutral and unbiased answer to a question, which is important for undertaking research. This is why the downstream Graph-Base RAG system, potentially designed for answering academic-related questions, needs a more global KG combined with multiple local KG.

12.3.3 Combining of KGs and LLMs

An interesting observation is that. In this study, we used LLMs to construct KGs from text documents. However, the constructed KGs can, in turn, help the LLM chatbots that answer user questions. There is a complement between LLMs and KGs, which has been highlighted in [Pan et al. \(2023\)](#)’s survey article. LLMs have great language ability. However, the way that LLMs are trained and inferred (as discussed in the background section 2.4) can make it hard to remember and explain structural facts with high accuracy. KGs, on the other hand, store every single piece of knowledge in a structured way. However, it does not have any language understanding, which requires someone or something with language understanding to extract and interpret the knowledge inside. Therefore, with the combination of LLMs and KGs, a more advanced AI system with both natural language ability and structural knowledge can be created ([Pan et al., 2023](#)).

12.4 Ethical Considerations

There are several potential ethical concerns associated with research in the areas of KGC, KGs, and LLMs, including those in the current research. Some of them can also become a study topic or future work.

12.4.1 Privacy Risks in KGC

We are building KG from an academic paper, and academic paper is generally non-confidential and publically available. However, if the sources are some personal data, such as diaries, the KG built will contain lots of personal information. For example, a person’s node may link to all their addresses, family members, workplace and many more. Since KG is structured, querying the graph makes it even easier than directly analysing the original unstructured document. Therefore, there is potentially an increase in the accessibility of private or sensitive information. Therefore, future research can focus on how to manage and regulate the construction of KGs as well as the encryption of KGs that contain sensitive information.

12 Concluding Remarks

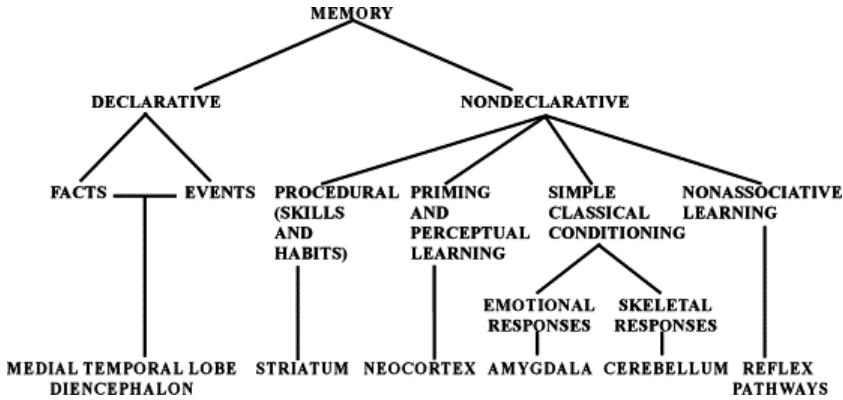


Figure 12.2: Types of Memory ([Squire, 2004](#))

12.4.2 Environmental Impact of KGC

As discussed in the background and evaluation chapters (Chapters 2 and 11), KGC pipelines, especially those based on LLM decoders, can have high runtime complexity since the complexity of LLM decoder calls is already high, which results in significant electrical energy consumption. However, not all electrical energy directly comes from renewable sources. Therefore, there is a potential for high carbon emissions, especially when KGC pipelines are used extensively.

12.4.3 Rapid Advancement of AIs

Two primary types of memory are involved in how a person learns ([Squire, 2004](#)), as shown in Figure 12.2. One is declarative memory, which stores explicit facts and knowledge ([Squire, 2004](#)). The other is nondeclarative memory, which includes procedural memory—commonly referred to as "muscle memory" ([Squire, 2004](#)).

For example, by practising a piece of music on a piano enough times, a person may learn how to play the music. However, this does not necessarily mean they remember every individual note on the sheet music or understand the musical theory behind it. They may just naturally play the next note based on the previous notes they have played.

Similarly, most LLMs these days operate by generating tokens based on what can potentially continue the previous tokens they generated or input by users. By training them on a sufficiently large text corpus, the LLMs can successfully continue many users' prompts, e.g., a question, to generate some plausible responses, which makes them seem to be knowledgeable. Yet, there is generally no sense of thinking but guessing. This is like playing the piano with muscle memory.

However, if KGs are embedded into LLMs, LLMs may start being able to think locally by querying the KGs they have. For example, they may check if a sentence generated aligns with the structural knowledge in the KGs. For any new or updated knowledge from the outer world, LLMs may also automatically build and update their KGs accordingly.

12.4 Ethical Considerations

Our research is exactly about how to enable LLMs to build KGs. However, if AIs in the future obtain both strong declarative and nondeclarative memory and are able to gather knowledge to improve themselves automatically, could they become too powerful?

Bibliography

- ASHOK, D. AND LIPTON, Z. C., 2023. Promptner: Prompting for named entity recognition. <https://arxiv.org/abs/2305.15444>. [Cited on pages 4, 23, 24, and 34.]
- AYOOLA, T.; TYAGI, S.; FISHER, J.; CHRISTODOULOPOULOS, C.; AND PIERLEONI, A., 2022. Refined: An efficient zero-shot-capable approach to end-to-end entity linking. <https://arxiv.org/abs/2207.04108>. [Cited on pages 33 and 94.]
- BANK, D.; KOENIGSTEIN, N.; AND GIRYES, R., 2020. Autoencoders. *arXiv e-prints*, (Mar. 2020), arXiv:2003.05991. doi:10.48550/arXiv.2003.05991. [Cited on page 133.]
- BRON, C. AND KERBOSCH, J., 1973. Algorithm 457: finding all cliques of an undirected graph. *Commun. ACM*, 16, 9 (Sep. 1973), 575–577. doi:10.1145/362342.362367. <https://doi.org/10.1145/362342.362367>. [Cited on page 86.]
- BROWN, T. B.; MANN, B.; RYDER, N.; SUBBIAH, M.; KAPLAN, J.; 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. M.; 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. <https://arxiv.org/abs/2005.14165>. [Cited on page 21.]
- DEVLIN, J.; CHANG, M.-W.; LEE, K.; AND TOUTANOVA, K., 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. <https://arxiv.org/abs/1810.04805>. [Cited on pages 26 and 33.]
- DING, L.; ZHOU, S.; XIAO, J.; AND HAN, J., 2024. Automated construction of theme-specific knowledge graphs. <https://arxiv.org/abs/2404.19146>. [Cited on pages 35, 36, and 43.]
- EDGE, D.; TRINH, H.; CHENG, N.; BRADLEY, J.; CHAO, A.; MODY, A.; TRUITT, S.; AND LARSON, J., 2024. From Local to Global: A Graph RAG Approach to Query-Focused Summarization. *arXiv e-prints*, (Apr. 2024), arXiv:2404.16130. doi:10.48550/arXiv.2404.16130. [Cited on page 138.]

Bibliography

- EDWARDS, A. AND CAMACHO-COLLADOS, J., 2024. Language models for text classification: Is in-context learning enough? <https://arxiv.org/abs/2403.17661>. [Cited on page 82.]
- ELMASRI, R. AND NAVATHE, S., 2015. *Fundamentals of Database Systems Seventh Edition*. Pearson. [Cited on pages 13 and 40.]
- HOGAN, A.; BLOMQVIST, E.; COCHEZ, M.; d'AMATO, C.; DE MELO, G.; GUTIERREZ, C.; LABRA GAYO, J. E.; KIRRANE, S.; NEUMAIER, S.; POLLERES, A.; NAVIGLI, R.; NGONGA NGOMO, A.-C.; RASHID, S. M.; RULA, A.; SCHMELZEISEN, L.; SEQUEDA, J.; STAAB, S.; AND ZIMMERMANN, A., 2020. Knowledge Graphs. *arXiv e-prints*, (Mar. 2020), arXiv:2003.02320. doi:10.48550/arXiv.2003.02320. [Cited on pages 1, 2, 3, 9, 11, 12, 13, 15, 23, 24, 25, 26, 40, 42, and 161.]
- INCE, M. N.; LEDET, J.; AND GUNAY, M., 2019. Building an open source linux computing system on risc-v. In *2019 1st International Informatics and Software Engineering Conference (UBMYK)*, 1–4. doi:10.1109/UBMYK48245.2019.8965559. [Cited on page 122.]
- JIA, R.; ZHANG, B.; MÉNDEZ, S. J. R.; AND OMRAN, P. G., 2024. Leveraging large language models for semantic query processing in a scholarly knowledge graph. <https://arxiv.org/abs/2405.15374>. [Cited on pages 6, 7, and 12.]
- JOSHI, M.; CHEN, D.; LIU, Y.; WELD, D. S.; ZETTLEMOYER, L.; AND LEVY, O., 2020. Spanbert: Improving pre-training by representing and predicting spans. <https://arxiv.org/abs/1907.10529>. [Cited on pages 30 and 94.]
- JOSHI, M.; LEVY, O.; ZETTLEMOYER, L.; AND WELD, D., 2019. BERT for coreference resolution: Baselines and analysis. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 5803–5808. Association for Computational Linguistics, Hong Kong, China. doi:10.18653/v1/D19-1588. <https://aclanthology.org/D19-1588>. [Cited on page 30.]
- LEE, K.; HE, L.; AND ZETTLEMOYER, L., 2018. Higher-order coreference resolution with coarse-to-fine inference. <https://arxiv.org/abs/1804.05392>. [Cited on page 33.]
- LI, B. Z.; MIN, S.; IYER, S.; MEHDAD, Y.; AND TAU YIH, W., 2020. Efficient one-pass end-to-end entity linking for questions. <https://arxiv.org/abs/2010.02413>. [Cited on pages 33 and 94.]
- LI, F.; LIN, Z.; ZHANG, M.; AND JI, D., 2021. A span-based model for joint overlapped and discontinuous named entity recognition. <https://arxiv.org/abs/2106.14373>. [Cited on pages 18 and 26.]

Bibliography

- LIU, Y.; HE, H.; HAN, T.; ZHANG, X.; LIU, M.; TIAN, J.; ZHANG, Y.; WANG, J.; GAO, X.; ZHONG, T.; PAN, Y.; XU, S.; WU, Z.; LIU, Z.; ZHANG, X.; ZHANG, S.; HU, X.; ZHANG, T.; QIANG, N.; LIU, T.; AND GE, B., 2024. Understanding llms: A comprehensive overview from training to inference. <https://arxiv.org/abs/2401.02038>. [Cited on pages 20 and 21.]
- LUAN, Y.; HE, L.; OSTENDORF, M.; AND HAJISHIRZI, H., 2018. Multi-task identification of entities, relations, and coreference for scientific knowledge graph construction. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, 3219–3232. Association for Computational Linguistics, Brussels, Belgium. doi:10.18653/v1/D18-1360. <https://aclanthology.org/D18-1360>. [Cited on pages 24, 49, 65, 122, and 145.]
- LYU, S. AND CHEN, H., 2021. Relation classification with entity type restriction. <https://arxiv.org/abs/2105.08393>. [Cited on page 33.]
- META, 2024. The llama 3 herd of models. <https://arxiv.org/abs/2407.21783>. [Cited on pages 3 and 33.]
- NOMIC, 2024. Gpt4all. <https://www.nomic.ai/gpt4all>. [Cited on page 48.]
- ONOE, Y.; BORATKO, M.; MCCALLUM, A.; AND DURRETT, G., 2021. Modeling fine-grained entity types with box embeddings. <https://arxiv.org/abs/2101.00345>. [Cited on page 118.]
- OPENAI, 2024a. Gpt-4 technical report. <https://arxiv.org/abs/2303.08774>. [Cited on pages 3 and 34.]
- OPENAI, 2024b. Learning to reason with llms. <https://openai.com/index/learning-to-reason-with-llms/>. [Cited on page 138.]
- OXFORD UNIVERSITY PRESS, 2024. Oxford learners dictionaries. <https://www.oxfordlearnersdictionaries.com/definition/english/entity>. [Cited on pages 1 and 15.]
- PAN, S.; LUO, L.; WANG, Y.; CHEN, C.; WANG, J.; AND WU, X., 2023. Unifying Large Language Models and Knowledge Graphs: A Roadmap. *arXiv e-prints*, (Jun. 2023), arXiv:2306.08302. doi:10.48550/arXiv.2306.08302. [Cited on pages 2, 3, 4, 6, 11, 17, 18, 21, 23, 24, 25, 26, 43, 45, 77, 93, 134, 138, 160, and 161.]
- PAN, S.; MÉNDEZ, S. J. R.; AND TAYLOR, K., 2022. A pipeline for analysing grant applications. <https://arxiv.org/abs/2210.16843>. [Cited on page 122.]
- PAPALUCA, A.; KREFL, D.; RODRIGUEZ, S. M.; LENSKY, A.; AND SUOMINEN, H., 2023. Zero- and few-shots knowledge graph triplet extraction with large language models. <https://arxiv.org/abs/2312.01954>. [Cited on pages 34, 35, 36, 42, and 43.]

Bibliography

- RADFORD, A.; WU, J.; CHILD, R.; LUAN, D.; AMODEI, D.; AND SUTSKEVER, I., 2019. Language models are unsupervised multitask learners. <https://api.semanticscholar.org/CorpusID:160025533>. [Cited on page 21.]
- ROBERTS, J., 2024. How powerful are decoder-only transformer neural models? In *2024 International Joint Conference on Neural Networks (IJCNN)*, vol. 1, 1–8. IEEE. doi:10.1109/ijcnn60899.2024.10651286. <http://dx.doi.org/10.1109/IJCNN60899.2024.10651286>. [Cited on pages 18, 19, and 21.]
- RODRÍGUEZ MÉNDEZ, S., 2018. Modeling actuations in bci-o: A context-based integration of sosa and iot-o. In *Proceedings of the 8th International Conference on the Internet of Things, IoT 2018*, ACM International Conference Proceeding Series. Association for Computing Machinery. doi:10.1145/3277593.3277914. Publisher Copyright: © 2018 Copyright is held by the owner/author(s). Publication rights licensed to ACM.; 8th International Conference on the Internet of Things, IoT 2018 ; Conference date: 15-10-2018 Through 18-10-2018. [Cited on page 121.]
- RODRÍGUEZ MÉNDEZ, S.; HALLER, A.; OMRAN, P.; WRIGHT, J.; AND TAYLOR, K., 2020. J2rm: an ontology-based json-to-rdf mapping tool. *CEUR Workshop Proceedings*, 2721 (2020), 369–373. Publisher Copyright: © 2020 CEUR-WS. All rights reserved.; 19th International Semantic Web Conference on Demos and Industry Tracks: From Novel Ideas to Industrial Practice, ISWC-Posters 2020 ; Conference date: 01-11-2020 Through 06-11-2020. [Cited on page 49.]
- RODRÍGUEZ MÉNDEZ, S.; OMRAN, P.; HALLER, A.; AND TAYLOR, K., 2021. Mel: Metadata extractor & loader. Publisher Copyright: © 2021 CEUR-WS. All rights reserved.; 2021 International Semantic Web Conference Posters, Demos and Industry Tracks: From Novel Ideas to Industrial Practice, ISWC-Posters-Demos-Industry 2021 ; Conference date: 24-10-2021 Through 28-10-2021. [Cited on page 121.]
- SAHOO, P.; SINGH, A. K.; SAHA, S.; JAIN, V.; MONDAL, S.; AND CHADHA, A., 2024. A Systematic Survey of Prompt Engineering in Large Language Models: Techniques and Applications. *arXiv e-prints*, (Feb. 2024), arXiv:2402.07927. doi:10.48550/arXiv.2402.07927. [Cited on pages 20 and 33.]
- SALMAN, M.; CHEN, H.; RODRÍGUEZ-MÉNDEZ, S. J.; AND HALLER, A., 2024. An LLM-SPARQL Hybrid Framework for Named Entity Linking and Disambiguation to Wikidata. [Cited on pages 33 and 94.]
- SENEVIRATNE, S.; RODRÍGUEZ MÉNDEZ, S. J.; ZHANG, X.; OMRAN, P. G.; TAYLOR, K.; AND HALLER, A., 2021. Tnnt: The named entity recognition toolkit. In *Proceedings of the 11th Knowledge Capture Conference, K-CAP '21* (Virtual Event, USA, 2021), 249–252. Association for Computing Machinery, New York, NY, USA. doi:10.1145/3460210.3493550. <https://doi.org/10.1145/3460210.3493550>. [Cited on page 122.]

Bibliography

- SHI, P. AND LIN, J., 2019. Simple bert models for relation extraction and semantic role labeling. <https://arxiv.org/abs/1904.05255>. [Cited on page 30.]
- SINGHAL, A., 2012. Introducing the knowledge graph: things, not strings. <https://blog.google/products/search/introducing-knowledge-graph-things-not/>. [Cited on page 2.]
- SQUIRE, L. R., 2004. Memory systems of the brain: A brief history and current perspective. *Neurobiology of Learning and Memory*, 82, 3 (2004), 171–177. doi: <https://doi.org/10.1016/j.nlm.2004.06.005>. <https://www.sciencedirect.com/science/article/pii/S1074742704000735>. Multiple Memory Systems. [Cited on page 162.]
- SUN, X.; LI, X.; LI, J.; WU, F.; GUO, S.; ZHANG, T.; AND WANG, G., 2023. Text classification via large language models. <https://arxiv.org/abs/2305.08377>. [Cited on page 82.]
- TANG, H.; CAO, Y.; ZHANG, Z.; CAO, J.; FANG, F.; WANG, S.; AND YIN, P., 2020. Hin: Hierarchical inference network for document-level relation extraction. <https://arxiv.org/abs/2003.12754>. [Cited on pages 33 and 43.]
- TARJAN, R., 1972. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1, 2 (1972), 146–160. doi:10.1137/0201010. <https://doi.org/10.1137/0201010>. [Cited on page 112.]
- VASWANI, A.; SHAZER, N.; PARMAR, N.; USZKOREIT, J.; JONES, L.; GOMEZ, A. N.; KAISER, L.; AND POLOSUKHIN, I., 2017. Attention Is All You Need. *arXiv e-prints*, (Jun. 2017), arXiv:1706.03762. doi:10.48550/arXiv.1706.03762. [Cited on pages 17 and 18.]
- W3C, 2014. Rdf 1.1 concepts and abstract syntax. <https://www.w3.org/TR/rdf11-concepts/>. [Cited on pages 11 and 12.]
- W3C, 2024. Rdf 1.2 concepts and abstract syntax. <https://www.w3.org/TR/rdf12-concepts/>. [Cited on pages 12 and 15.]
- WANG, D.; HU, W.; CAO, E.; AND SUN, W., 2020. Global-to-local neural networks for document-level relation extraction. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 3711–3721. Association for Computational Linguistics, Online. doi:10.18653/v1/2020.emnlp-main.303. <https://aclanthology.org/2020.emnlp-main.303>. [Cited on page 43.]
- WANG, H.; FOCKE, C.; SYLVESTER, R.; MISHRA, N.; AND WANG, W., 2019. Fine-tune bert for docred with two-step process. <https://arxiv.org/abs/1909.11898>. [Cited on page 43.]

Bibliography

- WANG, M.; LIU, Y.; LIANG, X.; LI, S.; HUANG, Y.; ZHANG, X.; SHEN, S.; GUAN, C.; WANG, D.; FENG, S.; ZHANG, H.; ZHANG, Y.; ZHENG, M.; AND ZHANG, C., 2024. Langgpt: Rethinking structured reusable prompt design framework for llms from the programming language. <https://arxiv.org/abs/2402.16929>. [Cited on page 84.]
- WANG, S.; SUN, X.; LI, X.; OUYANG, R.; WU, F.; ZHANG, T.; LI, J.; AND WANG, G., 2023. Gpt-ner: Named entity recognition via large language models. <https://arxiv.org/abs/2304.10428>. [Cited on pages 4, 23, 24, 28, 30, 34, 78, and 80.]
- WEI, X.; CUI, X.; CHENG, N.; WANG, X.; ZHANG, X.; HUANG, S.; XIE, P.; XU, J.; CHEN, Y.; ZHANG, M.; JIANG, Y.; AND HAN, W., 2024. Chatie: Zero-shot information extraction via chatting with chatgpt. <https://arxiv.org/abs/2302.10205>. [Cited on pages 30, 35, 36, and 43.]
- WIKIDATA, 2024. Wikidata main page. https://www.wikidata.org/wiki/Wikidata:Main_Page. [Cited on pages 12 and 41.]
- XIAO, S.; LIU, Z.; ZHANG, P.; MUENNIGHOFF, N.; LIAN, D.; AND NIE, J.-Y., 2024. C-pack: Packed resources for general chinese embeddings. <https://arxiv.org/abs/2309.07597>. [Cited on page 4.]
- XIE, T.; LI, Q.; ZHANG, J.; ZHANG, Y.; LIU, Z.; AND WANG, H., 2023. Empirical study of zero-shot ner with chatgpt. <https://arxiv.org/abs/2310.10035>. [Cited on pages 4, 23, 24, 28, 30, 34, and 80.]
- XUE, N.; WEISCHEDEL, R.; MARCUS, M.; PALMER, M.; HOVY, E.; BELVIN, R.; PRADHAN, S.; AND RAMSHAW, L., 2010. *OntoNotes: A Large Training Corpus for Enhanced Processing*. Springer. [Cited on page 41.]
- YAO, Y.; YE, D.; LI, P.; HAN, X.; LIN, Y.; LIU, Z.; LIU, Z.; HUANG, L.; ZHOU, J.; AND SUN, M., 2019. Docred: A large-scale document-level relation extraction dataset. <https://arxiv.org/abs/1906.06127>. [Cited on pages 24 and 49.]
- YE, D.; LIN, Y.; DU, J.; LIU, Z.; LI, P.; SUN, M.; AND LIU, Z., 2020. Coreferential reasoning learning for language representation. <https://arxiv.org/abs/2004.06870>. [Cited on page 23.]
- YE, H.; ZHANG, N.; CHEN, H.; AND CHEN, H., 2022. Generative Knowledge Graph Construction: A Review. *arXiv e-prints*, (Oct. 2022), arXiv:2210.12714. doi:10.48550/arXiv.2210.12714. [Cited on page 23.]
- YOU, H.; FU, Y.; WANG, Z.; YAZDANBAKHSH, A.; AND LIN, Y. C., 2024. When linear attention meets autoregressive decoding: Towards more effective and efficient linearized large language models. <https://arxiv.org/abs/2406.07368>. [Cited on pages 18 and 19.]

Bibliography

- ZHANG, B.; RODRÍGUEZ-MÉNDEZ, S.; AND OMRAN, P., 2023. Askg: An approach to enrich scholarly knowledge graphs through paper decomposition with deep learning. *CEUR Workshop Proceedings*, 3632 (2023). Publisher Copyright: © 2023 Copyright © 2023 for this paper by its authors.; 22nd International Semantic Web Conference on Posters, Demos and Industry Tracks: From Novel Ideas to Industrial Practice, ISWC-Posters-Demos-Industry 2023 ; Conference date: 06-11-2023 Through 10-11-2023. [Cited on pages 6 and 7.]
- ZHANG, B. AND SOH, H., 2024. Extract, define, canonicalize: An llm-based framework for knowledge graph construction. <https://arxiv.org/abs/2404.03868>. [Cited on pages 30, 36, 42, and 43.]
- ZHENG, J. AND CHEN, Z., 2023. Sentence-level relation extraction via contrastive learning with descriptive relation prompts. <https://arxiv.org/abs/2304.04935>. [Cited on page 33.]
- ZHONG, L.; WU, J.; LI, Q.; PENG, H.; AND WU, X., 2023. A comprehensive survey on automatic knowledge graph construction. *ACM Comput. Surv.*, 56, 4 (Nov. 2023). doi:10.1145/3618295. <https://doi.org/10.1145/3618295>. [Cited on pages 23, 24, and 44.]
- ZHOU, Z.; NING, X.; HONG, K.; FU, T.; XU, J.; LI, S.; LOU, Y.; WANG, L.; YUAN, Z.; LI, X.; YAN, S.; DAI, G.; ZHANG, X.-P.; DONG, Y.; AND WANG, Y., 2024. A survey on efficient inference for large language models. <https://arxiv.org/abs/2404.14294>. [Cited on page 18.]