

ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI BOLOGNA

DEPARTMENT OF COMPUTER SCIENCE AND
ENGINEERING

Master Program in Artificial Intelligence

**SUBGRAPH RETRIEVAL FOR BIOMEDICAL OPEN-DOMAIN
QUESTION ANSWERING: UNLOCKING THE KNOWLEDGE
GRAPH EMBEDDING POWER**

Master Thesis in
Data Mining, Text Mining, and Big Data Analytics

Supervisor

Prof. Claudio Sartori

Candidate

Faisal Ramzan

Co-relatori

Prof. Gianluca Moro

Dr. Giacomo Frisoni

Third Session
Academic Year 2021 – 2022

KEY WORDS

Natural Language Processing
Open-domain Questioning Answering
Knowledge Graphs
Subgraph Retrieval
Multi-hop Reasoning

Seek knowledge from the Cradle to the Grave.
- Prophet Muhammad, Al-Hadith

*Education is the key to unlocking the world,
a passport to freedom.*
- Oprah Winfrey

*Education is the passport to the future,
for tomorrow belongs to those who prepare for it today.*
- Malcolm X

Abstract

The primary purpose of question answering is to identify the relevant portion of knowledge from the data corpus and be able to reason over it to extract the correct answer to the given question. The available knowledge can be encoded implicitly in large pre-trained language models (LMs) on unstructured text (e.g., BERT, RoBERTa) or explicitly in structured knowledge graphs (KGs), such as Freebase and ConceptNet, where entities are represented as nodes and relations between them as edges. Structured KG is more popular than KG; Language Models do not capture the semantic meaning of the same context with billions of parameters. While retrieving the entire Knowledge Graph is quite challenging concerning the size and memory issues. Moreover, inferring the answer to the question takes time during the reasoning on the whole KG, affecting the reasoning phase, which causes finding the incorrect solution. Pre-trained LMs have broad knowledge coverage but must perform better on structured reasoning, such as handling negation and flipped conditions. We aim to retrieve the relevant portion of the subgraph from the large KG graph. The existing subgraph retrieval solutions primarily focus on discriminative k-hop approaches or SPARQL queries on massive KGs. However, they require time-consuming, unsustainable operations in real-world contexts like biomedicine, where entities and known relationships among them are massive. They frequently rely on Named entity-linking NEL tools that fail in recognizing and mapping entities without being capable of generalizing to similar or high-order concepts. Instead, approximated search on dense representations of KGs and text can significantly boost the effectiveness and efficiency of subgraph construction with the help of enhanced generalization capabilities that overcome NEL limits and the possibility of indexing embeddings and speeding up top-K retrieval operations. In our work, we analyzed the existing methods of subgraph construction. However, they could be more efficient because of their size and quality of retrieved subgraph, which affect the reasoning process for extracting an answer to the question. Therefore, we propose the Subgraph Retrieval that tries to find the more relevant entities through linked paths (path queries) to the topic entities. The goal is to find the sequence of relations or paths and their connected entities linked to the topic entities by measuring their similarity

between them in the dense space setting.

Acknowledgments

At the end of this course of study, it is appropriate and the right moment to dedicate a few lines of thanks to all who have accompanied me along this adventure.

First, I would like to express my deepest gratitude to my supervisor Claudio Sartori, who guided me step by step in realizing this work. I will always be grateful for his care, availability, and competence. His experience and professionalism are fundamental.

At the same time, I would like to thank my co-supervisor, Prof. Gianluca Moro, and Mr. Dr. Giacomo Frisoni, for their kindness, courtesy, and insight. Giacomo led me to consider continuing my journey into the world of research.

Special thanks to my family members for their love and support, for their teachings, and for tempering my character. They are the ones who have always instilled in me the awareness that optimal results can be achieved only through commitment and passion. I want to thank my whole family for the unconditional love they show me every day, even when everything seems complicated and unreachable.

I cannot forget to thank my friends for their complicity, the laughter, the anguish we have always shared, and for their unfailing pats on the back.

Many thanks to Jyoti, Zhoalb, and Ehtsham for being best friends. They listen to me without judging and are always present even when absent; I appreciate their loyalty and simplicity.

Faisal Ramzan
01 March, 2023

Introduction

The question and Answering (QA) system aims to provide users with accurate and relevant answers to their questions through automated means. This can be achieved through various techniques such as natural language processing, information retrieval, and machine learning. The question-answering system needs high-level reasoning to extract the answer from the text’s retrieved portion or from the retrieved subgraph. We can have two classifications of question-answering: (1) Open-domain question answering is very challenging because there needs to be evidence or context given to the question. The existing proposed work retrieves the context or evidence using different methods for retrieving text from documents through TF-IDF and BM25 models. For example, the exam modality of a student by considering the open and closed book setting. Open-book questions permit using reference materials, such as textbooks, notes, or other resources, during the exam. The goal of an open book is to understand and apply information rather than just memorization. Therefore, open-book exams require more critical reasoning and problem-solving skills.

(2) The closed-book questions do not allow using any reference materials during the exam. This is because closed-book exams aim to test a student’s recall and memorization of information. Instead, the close book domain deals with questions related to a specific domain (like medicine) and can leverage domain-specific knowledge often formalized in ontologies. An integral part of question answering is identifying the relevant portion of knowledge and being able to reason over it. Knowledge can be encoded implicitly in large pre-trained language models (LMs) on unstructured text (e.g., BERT, RoBERTa) or explicitly in structured knowledge graphs (KGs), such as Freebase and ConceptNet, where entities are represented as nodes and relations between them as edges.

Pre-trained LMs have broad knowledge coverage but must perform better on structured reasoning, such as handling negation and flipped conditions. On the other hand, KGs are suited to structured reasoning and enable explainable predictions. However, they may need more coverage and be noisy because of the size and quality of the retrieved relevant subgraph. To perform reasoning effectively on both sources of knowledge remains a significant open problem. We

aim to retrieve the relevant portion of the subgraph from the large graph. The existing subgraph retrieval solutions primarily focus on discriminative k-hop approaches or SPARQL queries on massive KGs. However, they require time-consuming, unsustainable operations in real-world contexts like biomedicine, where entities and known relationships among them are massive. They frequently rely on Named entity-linking NEL tools that fail in recognizing and mapping entities without being capable of generalizing to similar or high-order concepts. Instead, approximated search on dense representations of KGs and text can significantly boost the effectiveness and efficiency of subgraph construction with the help of enhanced generalization capabilities that overcome NEL limits and the possibility of indexing embeddings and speeding up top-K retrieval operations.

In our work, we analyzed the existing methods of subgraph construction. However, they are less efficient because of their size and quality of retrieved subgraph, which affect the reasoning process for extracting an answer to the question. Therefore, we propose the decoupled Subgraph Retrieval that tries to find the more relevant entities through linked paths (path queries) to the topic entities. Our search starts from the topic entities to expand and travel the relevant path and predict the tails nodes. After expanding all relevant paths to the topic entities, we merge the expanded tree by taking the union of all expanded paths and then merging them into a standard unified graph. The goal is to find the sequence of relations or paths and their connected entities linked to the topic entities by measuring their similarity between them. In this thesis work, we deal with biomedical data UMLS that belongs to the MedQA-USMLE dataset, which contains multiple biomedical domain exam questions and answers featured in the United States.

Motivation

Combining a language model with a knowledge graph enhances reasoning capabilities by providing the language model with structured, multi-relational, and context-aware knowledge that can be used to answer questions, generate explanations, and support decision-making. For example, the language model can use the knowledge graph to perform inferences, generate hypotheses, and resolve vague references like negations and flipped constraints in the question. In contrast, the knowledge graph can leverage the language model’s ability to process and generate natural language text. This synergy leads to a system that can perform complex reasoning tasks while communicating results in a way understandable to humans.

Contribution

Integrating language models and knowledge graphs can enhance reasoning abilities in AI systems. Combining language models' natural language processing capabilities with the structured representation of knowledge in a graph allows these systems to perform tasks such as question answering and relationship extraction with greater accuracy and efficiency. Furthermore, this combination can also allow for the inference of new knowledge by combining and extrapolating information from the language model and knowledge graph. Overall, the integration of these two technologies has the potential to significantly improve the ability of AI systems to perform human-like reasoning.

Social Impact

The social impact of combining a language model with a knowledge graph for reasoning can be more positive than negative. On the positive side, it has the potential to improve decision-making and increase efficiency in various industries, such as healthcare, finance, and education. It can also provide access to knowledge and information to individuals who might not otherwise have it.

On the negative side, there is a potential for misuse of the technology, for example, in the spread of misinformation, biased decision-making, and the reinforcement of existing inequalities. There is also a concern about privacy, as using such systems may result in collecting and exploiting sensitive personal information. Additionally, developing and deploying these systems must be done ethically and responsibly, considering the potential impacts on society and addressing potential biases in the training data and algorithms. Developing robust and transparent evaluation methods is also essential to ensure that the technology is being used to benefit society.

Thesis Organization

- **Chapter 1** - Briefly discusses the theoretical background on the contribution of natural language processing (NLP) transformers, language models, and knowledge graphs.
- **Chapter 2** - Analyzing the existing information retrieval phase and discussing the challenge of the existing studies because they have some limitations related to the size of the retrieved subgraph and incomplete subgraph because of the weak internal intermediate node or entities also information extraction from different sources.

- **Chapter 3** - In this chapter, we propose our solution and how it differs from the existing proposed solution with complete graphical illustrations. It also presents some aspects of the implementation of Subgraph retrieval, expressing different properties of Knowledge Graphs, graph embeddings, and types, and the dataset used in the experimental phase.
- **Chapter 4** - In this chapter, we implement the solution proposed in chapter 03. we start from the initial preprocessing of the dataset, construction of triple stores, implementation of different knowledge graph embedding techniques, embedding exploration, expressing the KG data and designing SPARQL queries for data collection, focusing on model fine-tuning and graph merging techniques, and expressing the results of experiments.
- **Chapter 5** - In this chapter, we analyze the results of our experiments during the studies and demonstrate their comparison concerning existing proposed solutions. Also, how the results differ from the existing solution and how to devise the subgraph retrieval phase into other similar work like QAGNN, Dragon, and GreaseLM considered the future direction.

Index

| | | |
|----------|--|-----------|
| 1 | Theoretical Framework | 1 |
| 1.1 | Recent Breakthrough in Natural Language Processing | 1 |
| 1.1.1 | Importance of Unstructured data | 2 |
| 1.1.2 | Structured Information Extraction from Complex Scientific Text with Fine-tuned Large Language Models | 2 |
| 1.1.3 | Information Extraction From Textual data | 3 |
| 1.1.4 | Named Entity Recognition and Relation Extraction | 4 |
| 1.1.5 | Language Models and Transformers | 5 |
| 1.1.6 | Google Bi-directional Encoder Representations from Transformers | 7 |
| 1.1.7 | Word Embedding Methods | 9 |
| 1.1.8 | Language Models for Open and Closed-domain Question Answering | 12 |
| 1.2 | Knowledge Graphs | 14 |
| 1.2.1 | Preliminary Definitions of Graphs | 14 |
| 1.2.2 | Graph Types and their Properties | 14 |
| 1.2.3 | Graph Embedding Techniques | 15 |
| 1.2.4 | Example of Knowledge Graphs | 20 |
| 1.3 | Language Models and Structured Knowledge Graph | 23 |
| 1.3.1 | Limitations of Pure Language Models | 23 |
| 1.3.2 | Benefits of Combining Language Models and Knowledge Graphs | 24 |
| 2 | Related Work | 27 |
| 2.1 | Knowledge-based Data (Triplets) | 27 |
| 2.2 | Reasoning with Language Models and Knowledge Graphs | 29 |
| 2.2.1 | QA-GNN | 29 |
| 2.2.2 | GreaseLM: Graph Reasoning Enhanced Language Models For Question Answering | 33 |
| 2.2.3 | Dragon: Deep Bi-directional Language Knowledge Graph Pre-training | 36 |

| | | |
|----------|--|------------|
| 2.2.4 | UNIKGQA: Unified Retrieval and Reasoning For Solving Multi-Hop Question Answering Over Knowledge Graph . | 39 |
| 2.2.5 | CODER: Knowledge Infused Cross-Lingual Medical Term Embedding For Term Normalization | 43 |
| 2.3 | Challenges and Limitations in the Existing Subgraph Retrieval Phase | 47 |
| 2.4 | Dense Retrieval | 50 |
| 2.4.1 | Language Model for Dense Passage Retrieval | 52 |
| 2.4.2 | Differentiate between Single-hop and Multi-hop reasoning | 53 |
| 3 | Our Solution | 55 |
| 3.1 | Stanford UMLS Knowledge Graph | 55 |
| 3.2 | Knowledge Graph Embeddings | 57 |
| 3.3 | Reasoning over Knowledge Graphs | 58 |
| 3.3.1 | Subgraph Retrieval Enhanced Model for Multi-hop Knowledge Base Question Answering | 60 |
| 3.3.2 | How to expand paths from topic entities? | 62 |
| 3.3.3 | Subgraph Construction Through Expanded Paths | 64 |
| 3.3.4 | Subgraph Retrieval and Training | 66 |
| 4 | Experimental setup | 69 |
| 4.1 | Dataset | 69 |
| 4.2 | Baseline Model | 77 |
| 4.3 | Data Collection Methods | 80 |
| 4.3.1 | Expressing Knowledge Graph Properties Through NetworkX | 80 |
| 4.3.2 | GraphDB | 83 |
| 4.4 | Graph Merging | 87 |
| 5 | Results | 93 |
| 5.0.1 | Knowledge Graph Embedding Models Performance Evaluation | 96 |
| 5.0.2 | Training a CODER model | 99 |
| | Bibliografia | 105 |

List of Figures

| | | |
|------|---|----|
| 1.1 | Given figures indicates the importance of Structured Vs. Unstructured by showing their comparison. | 3 |
| 1.2 | Given image demonstrates how Information Extraction systems extract the structured output sequences from unstructured web text. | 4 |
| 1.3 | How NER and RE work and produces the output taken from [1]. | 5 |
| 1.4 | The Transformer Model Architecture picture taken from paper “ Attention Is All You Need. ” [2] | 6 |
| 1.5 | The classic example of a BERT-based sentiment analysis system for product reviews is whether the users’ feedback about the product is positive or negative. | 7 |
| 1.6 | Google BERT Architecture [3] | 8 |
| 1.7 | Framework of BioGPT when adapting to downstream tasks taken from [4]. | 9 |
| 1.8 | Similar words placed near to each other in the space. | 11 |
| 1.9 | Trajectories of brand names and people through time: Apple, Amazon, Obama, and Trump, Image taken from [5]. | 11 |
| 1.10 | Explains the concept of open and closed-domain question-answering setup. | 13 |
| 1.11 | Graphical representation of graphs from the left: un-directed graph, directed graph, un-directed multi-graph, and directed multi-graph. | 16 |
| 1.12 | How Deepwalk traverses the nodes, Image taken from [6]. | 17 |
| 1.13 | Representation of TransE and TransH Models image taken from [7] | 19 |
| 1.14 | Example of Google Knowledge Graph | 21 |
| 1.15 | Example of world’s largest DBpedia Knowledge Graph | 22 |
| 2.1 | Structural Representation of the Knowledge Base and the Knowledge Graph. | 28 |

| | | |
|------|---|----|
| 2.2 | Overview of their approach. Given a QA context (z), they connect it with the retrieved KG to form a joint graph, compute the relevance of each KG node conditioned, and perform reasoning on the working graph | 30 |
| 2.3 | Relevance scoring of the retrieved KG: they use a pre-trained LM to calculate the relevance of each KG entity node conditioned on the QA context. | 31 |
| 2.4 | Test accuracy on MedQA-USMLE | 32 |
| 2.5 | General architecture of GreaseLM taken from [8] | 35 |
| 2.6 | Qualitative analysis of GREASELM’s graph attention weight changes across multiple messages passing layers compared with QA-GNN. GREASELM demonstrates attention change patterns that more closely resemble the expected change in focus on the “bug” entity.[8] | 36 |
| 2.7 | Performance on MedQA-USMLE take from [8], demonstrate that GreaseLM outperforms state-of-the-art fine-tuned LMs (e.g., SapBERT) and a QA-GNN augmentation of SapBERT. | 37 |
| 2.8 | General architecture of DRAGON taken from [9] | 37 |
| 2.9 | Performance on the 9 downstream commonsense reasoning tasks. [9] | 39 |
| 2.10 | Accuracy on biomedical NLP tasks. DRAGON outperforms all previous biomedical LMs. | 39 |
| 2.11 | Illustrative examples for our work: (a) an example of multi-hop KGQA; (b) an example of abstract subgraph; and (c) the overall learning procedure of our UniKGQA model [10]. | 41 |
| 2.12 | The overview of the unified model architecture of UniKGQA, consisting of two modules, i.e., semantic matching and matching information propagation ([10]). | 42 |
| 2.13 | Performance comparison of different methods for KGQA (Hits@1 and F1 in percent). We copy the results for TransferNet and others. Bold and underlined fonts denote the best, and the second-best methods [10]. | 43 |
| 2.14 | Positive and Negative labels pairs, image taken from [11]. | 44 |
| 2.15 | The overview of CODER. CODER encodes terms potentially in different languages into the embedding space. Term-term similarities and term-relation-term similarities are calculated to train CODER. | 45 |
| 2.16 | Acc@k for different embeddings in Cadec and Psy-Tar datasets. Contextual embeddings report results using the average pooling representation. | 48 |

| | | |
|------|--|-----|
| 2.17 | The weak or missing edges between the nodes image taken from [9]. | 50 |
| 2.18 | Information Retrieval and Reasoner phase | 51 |
| 2.19 | Graphical representation of the equation 2.3,How to measure the similarities between the embedding of question E_Q and the passage E_P | 53 |
| 2.20 | Difference between single and multi-hop reasoning for inferring answer to the given question. | 54 |
| 3.1 | Example of Bio-Medical Knowledge Graph (UMLS). | 56 |
| 3.2 | Illustration of the subgraph retrieving process. We expand a path from each topic entity, induce a corresponding tree, and then merge the trees from different topic entities to form a unified subgraph [12]. | 62 |
| 3.3 | Trees inducing through the selected paths as shown in the image, taken from the work [13]. | 65 |
| 3.4 | Trees merging into a unified common graph taken from | 65 |
| 3.5 | Subgraph Retrieval SR and training phases. | 66 |
| 4.1 | Google Colab notebook resources utilizations. | 70 |
| 4.2 | 2 Figures side by side | 75 |
| 4.3 | Link prediction results: for the 5 KGE models on SNOMED-CT (top); and TransE and RotatE on two standard KGE datasets. | 77 |
| 4.4 | Directed graph with Networkx and Matplotlib. | 81 |
| 4.5 | Data Transformation through OntoText Refine. | 84 |
| 5.1 | GraphDB: SPARQL query execution time for retrieving particular relations. | 93 |
| 5.2 | GraphDB: SPARQL query execution time to retrieve the tail node of the relations. | 94 |
| 5.3 | GraphDB: To execute a query on a graph database to retrieve relations from knowledge graph triples, | 95 |
| 5.4 | Coder: Training Loss | 100 |

List of Tables

| | | |
|-----|---|----|
| 4.1 | Example of Stanford UMLS KG Triples | 71 |
| 4.2 | Unique UMLS Knowledge Graph features (Head, Tail, and Relations). | 72 |
| 5.1 | Performance Metric on UMLS KG | 97 |
| 5.2 | Tail Prediction using DistMult | 97 |
| 5.3 | Tail Prediction using TransE | 98 |

Chapter 1

Theoretical Framework

In this chapter, we will see some of the essential preliminary studies about Language Models LMs (**Section 1.1**) and Knowledge Graphs KGs (**Section 1.2**) considered as the backbone of this study. The work proposed by Yasunaga [14], explains why communities combine them (**Section 1.3**) to perform joint reasoning and predict the answer to the given questions. Our focus is retrieving the relevant portion of a domain-specific Knowledge Graph.

1.1 Recent Breakthrough in Natural Language Processing

Suppose we have an input text: "**Faisal is a person who belongs to Pakistan. He is a student of a master's degree, and his major is computer science**". Humans can easily recognize the meaning of words like "Faisal," "He," and "His," but for the machine, it is challenging without supervision. Still, it is difficult for neural networks or natural language understanding systems to understand the given meaning of the mentioned entities, their relationships, and the dependencies between the words. Modern Natural Language Processing (NLP) relies on transformers-based language models and structured knowledge graphs to overcome the problem of semantics and dependencies.

In advance of NLP, several methods deal with different natural language understanding tasks. For example, the initial implementation of **Sequence-to-sequence (seq2seq)**[15] and **Recurrent Neural Network (RNN)** ([16]) models had outstanding achievements in the NLP tasks like Machine Translation, Text Summarization, Speech Recognition, Question-Answering,

and so on. However, these models have some limitations. For example, Seq2seq models fail to deal with long-range dependencies, train, and process them; the long-input sequences do not allow parallelization, and the computation of the neural network is very slow, which can face problems like Exploding or Gradient Vanishing.

1.1.1 Importance of Unstructured data

Unstructured data are data types that do not follow or are not defined through any models or schemas and have no semantic meanings. Such data include web pages, emails, blogs, legal and medical document repositories, etc. Still, it is essential data, and many businesses work on it to make it helpful information. Through past surveys, we obtain the importance of unstructured data. The Butler Group can explain the increasing importance of unstructured data.

"85% of all data stored is held in an unstructured format."

According to the Gartner Group:

First: *"80% of business is conducted on unstructured information."*

Second: *"Unstructured data doubles every three months."*

According to the survey of 1996, Figure 1.1a explains the dimension and the **market cap of the structured vs. unstructured data**. The histogram of both sources of knowledge can clearly show that in 1996, most businesses spent their maximum budgets on the structured source of information, while in the survey of 2009, as shown in Figure 1.1b, the most famous companies like **Google, Yahoo, and Bing, etc.** spend their maximum budget on the unstructured data.

Natural Language Processing and Machine Learning methods combined to process the raw data and preserve their semantic meanings. **Information extraction (IE)** methods extract structured information from unstructured data like **entities, relations, objects, events**, and many other types.

1.1.2 Structured Information Extraction from Complex Scientific Text with Fine-tuned Large Language Models

The information extraction and establishing the link with the complex scientific information from unstructured text is a challenging task, particularly



Figure 1.1: Given figures indicates the importance of **Structured Vs. Unstructured** by showing their comparison.

for those who don't know much about natural language processing and machine learning. In this work, we have an idea of a sequence-to-sequence approach that joins the **Named Entity Recognition** and **Relation Extraction** to process the complex hierarchical data.

This methodology is based on the sizeable **Large pre-trained Language Model (LLM)** [17] like **Generative Pre-trained Transformer GPT-3** [17], **BERT** [3]. The chat GPT-3 model is fine-tuned on many wikis and other open and closed domain documents. All the information about the different domains can be encoded into the language model, which is used for self-supervised tasks such as classifying the sentiments and the following sentence predictions (Next Sentence Prediction).

In these related works [1, 17], the extraction of the information can be from documents, single passage, or a single sentence or multiple sentences, and after using the NLP techniques like name entity recognition NER [18] and abstract meaning representations AMR [19] that helps to capture the semantics of each work and their relations.

1.1.3 Information Extraction From Textual data

Researchers from different domains are trying to find efficient techniques to analyze better and interpret human input data, such as human voices or handwriting. There are several tasks of Machine translation, NLU, QA, and many of the Information retrieval and information extraction systems used to resolve the daily high-level tasks as shown in Figure 1.2.

The primary purpose of **information extraction systems (IE)** is to extract structured information from unstructured data and represent it in a format that is easy to process and understandable by the end users [20]. The **IE** system extracts the set of instances, like **functional categories, entities, relations, etc.**, from the unambiguous representation. It organizes them into

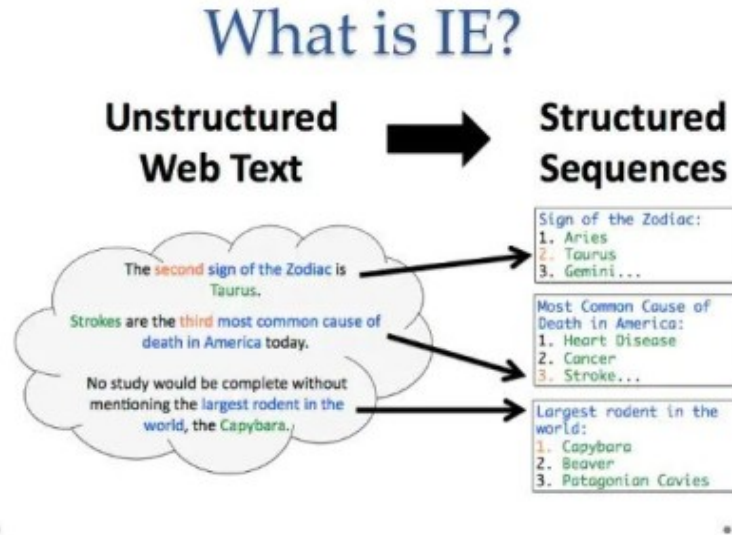


Figure 1.2: Given image demonstrates how Information Extraction systems extract the structured output sequences from unstructured web text.

a structured or knowledge-based format for making decisions.

The system takes a set of documents as inputs and generates a representation of relevant documents concerning the given inputs. IE systems aim to identify the salient feature from the input text and organize them according to different knowledge sources like Databases, Data Mart, Language Models, Knowledge Graphs, etc. ([21]).

1.1.4 Named Entity Recognition and Relation Extraction

The **Named Entity Recognition (NER)** applied to the input text helps us to find the entities mentioned in the given text, such as some text about a location like Rome, Paris, etc. Likewise, persons like Barack Obama, Steve Hopkins, etc., are identified by using this approach. For example, Figure 1.3 shows the result of NER [1] techniques. It identifies some operated entities such as disease, drugs, chemicals, proteins, etc. In this process, entities and relations are recognized and semantically classified into their relevant domains.

The famous applications where the NER concept is used are question answering, machine translation, automatic text summarization, text mining, information retrieval, and opinion mining. The higher efficiency and accuracy of these systems are significant, but dealing with big data brings new challenges

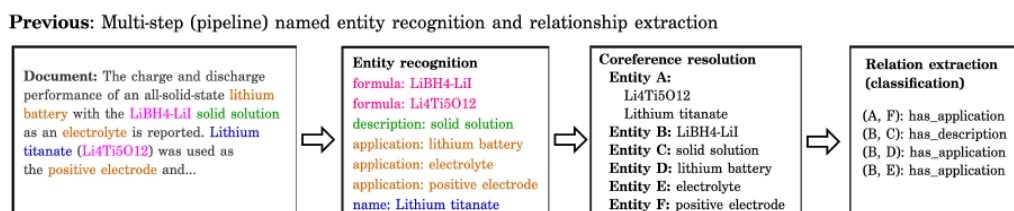


Figure 1.3: How **NER** and **RE** work and produces the output taken from [1].

to these systems. These challenges are volume, variety, and velocity of the data; applying **NER** on the text corpora will remove all of the irrelevant details and make it more efficient than the existing one.

Relation Extraction RE is used to extract the relationship between the entities. It analyzes the semantic relationship between the entities, including lexical, syntactic, and morphological, and focuses on the data contextual properties. The RE extracts the relation between the entities and represents what type of relationship between the entities exists, like one-to-one, one-to-many, etc.

1.1.5 Language Models and Transformers

The concept of Transformers in NLP was proposed in the article “**Attention Is All You Need**” [2] that aims to resolve the current issues of seq2seq models with the handling of long-range input sequences and the dependencies between the words and sentences. Introducing NLP transformers and Language models makes it easy to capture the relationships in the given sequence of words and deal with long sequences of inputs.

The famous quotation of this paper:

“The Transformer is the first transduction (transduction means the conversion of the input sequence to the output sequence) model relying entirely on self-attention to compute representations of its input and output without using sequence-aligned RNNs or convolution.”

Initial experiments on two machine translation tasks prove these models to be superior and efficient in quality. At the same time, it is also more parallelizable and requires significantly very less training time. Implementing a transformer is to completely handle the dependencies between input and output with self-attention and recurrence.

Transformer Model architecture In the transformer model architecture, the left part is the encoder of the model that maps an input sequence of symbol representations (x_1, x_2, \dots, x_n) to a sequence of continuous representations $z = (z_1, z_2, \dots, z_n)$. Given the z representations, the right part of the model decoder generates an output sequence (y_1, y_2, \dots, y_m) of symbols one element at a time. At each step, the model is auto-regressive, consuming the previously generated symbols as additional input when generating the next.

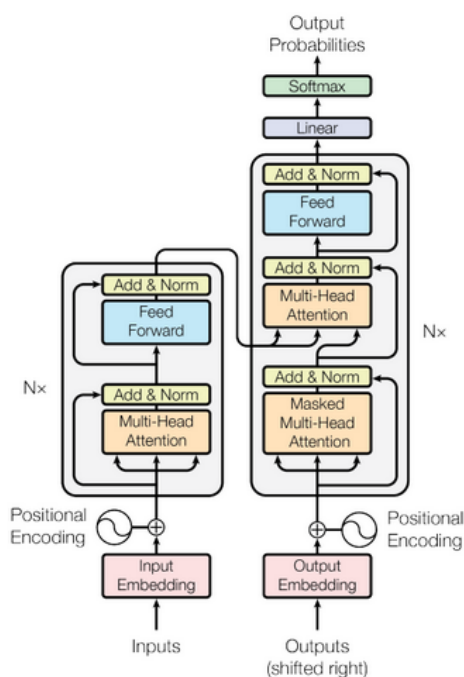


Figure 1.4: The Transformer Model Architecture picture taken from paper “**Attention Is All You Need.**” [2]

The transformer model uses a stacked self-attention mechanism in the encoder and decoder. The encoder takes the input sequence and produces a set of hidden states passed to the decoder part. The decoder uses these hidden states and self-attention layers to generate the output sequence. In addition to the self-attention layers, the encoder, and decoder include point-wise, fully connected layers. This architecture allows the model to effectively process sequences of varying lengths and generate output sequences conditioned on the input, as shown in Figure 1.4.

1.1.6 Google Bi-directional Encoder Representations from Transformers

Bidirectional Encoder Representations from Transformers [3] a language representation model developed by Google AI. It uses pre-training and fine-tuning phases to create state-of-the-art models for various tasks. These tasks include question-answering systems, sentiment analysis, and language inference. The main goal of the BERT-based LMs [3] is to help the information retrieval systems better understand the context around your searches. For example, Figure 1.5 shows the sentiment classification task, which represents the sentiments of the user reviews, either positive or negative, on the base of the computation by BERT LM.

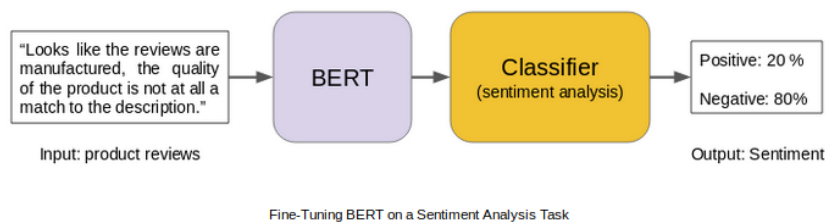


Figure 1.5: The classic example of a BERT-based sentiment analysis system for product reviews is whether the users' feedback about the product is positive or negative.

The architecture of the BERT in Figure 1.7, a multi-layer bidirectional Transformer encoder, and the self-attention layer perform self-attention in both directions forward and backward. The variants of Google BERT are shown below in Figure 1.7, with the number of transformer layers and parameters used for tuning the model.

1. BERT Base: Number of Transformers layers = 12, Total Parameters = 110M
2. BERT Large: Number of Transformers layers = 24, Total Parameters = 340M

The language models like Google BERT [3], and RoBERTa [22] are trained on a large corpus of text data. As a result, it can be fine-tuned for various natural language processing tasks, such as question answering, sentiment analysis, and named entity recognition. One of the key innovations of BERT is its use of "**Masked Language Modeling**" during pre-training, where a percentage of the words in the input is replaced or masked with a unique token, and the

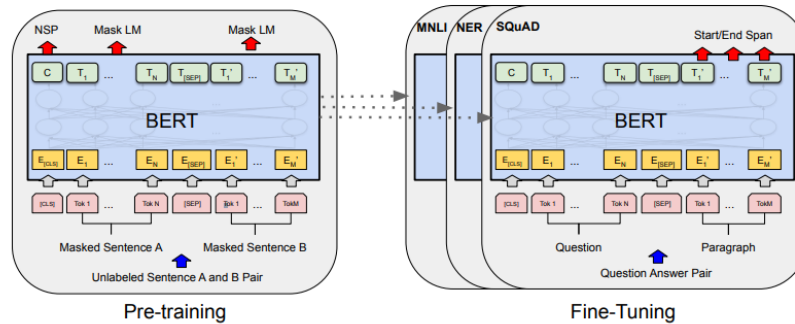


Figure 1.6: Google BERT Architecture [3]

model is trained to predict the original word based on its context. This allows BERT to learn **context-dependent representations** of words, which is particularly useful for tasks like question answering, where understanding the context is crucial.

Overall the pre-training and fine-tuning BERT is shown in Figure 1.7. Apart from output layers, the same architectures are used in pre-training and fine-tuning. In most classification problems, we use one part of the model, like the encoder, for classification purposes. The same pre-trained model parameters are used to initialize models for different downstream tasks. To fine-tune the model, we need to use the pre-trained or frozen model, add a new layer on top of the existing one, and perform classification tasks, like sentiment analysis, to classify positive and negative sentiments of product reviews and many other tasks.

BioGPT

BioGPT [4] is a domain-specific language model pre-trained on large-scale biomedical literature. It is a type of Generative Transformer Language Model, which means that it has been trained on a large corpus of data, and the purpose is to generate text based on those patterns learned during the training on the data. BioGPT aims to provide a well-suited model for generating and processing natural language text related to biomedicine.

By being pre-trained on a large amount of biomedical data, BioGPT can perform a broad range of natural language processing tasks related to bio-medicine, including information extraction, question answering, and text summarization. Furthermore, this pre-training allows the model to perform these tasks more effectively and accurately than models not pre-trained on such a specialized dataset. Overall, BioGPT represents a promising development in domain-specific language models and has the potential to provide significant advances

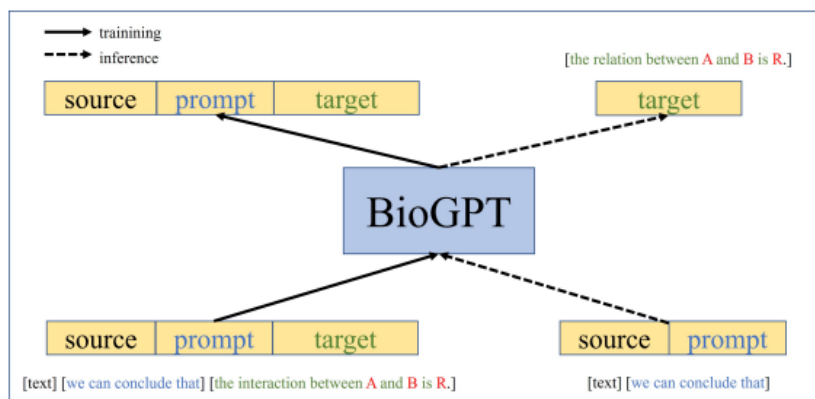


Figure 1.7: Framework of BioGPT when adapting to downstream tasks taken from [4].

in various areas of bio-medicine.

BioGPT, a domain-specific generative Transformer language model pre-trained on the large-scale biomedical corpus. BioGPT tested on six biomedical NLP tasks and demonstrated that our model outperforms previous models on most tasks. Primarily, we get 44.98%, 38.42%, and 40.76% F1 scores on BC5CDR, KD-DTI, and DDI end-to-end relation extraction tasks, respectively, and 78.2% accuracy on PubMedQA, creating a new record. Our larger model BioGPT-Large achieves 81.0% on PubMedQA.

1.1.7 Word Embedding Methods

This section will explain embedding and how algorithms deal with textual information. We aim to extract useful information and features from the given input in NLP. The input is in the human language, which consists of words, sentences, and documents in different languages like **English, Hindi, and Chinese**.

General natural language processing tasks are followed as:

- Text summarization: extractive or abstractive text summarization.
- Sentiment Analysis [Positive, Negative].
- Neural Machine Translation (NMT), Translating the text from one language to another.
- GPT2 [23] , GPT3 [24] Chatbots.

For processing the input information, we need to transform in such a representation which is helpful for algorithms because machine and deep learning algorithms only process the numeric input. So we need to find a mechanism that changes the information into numerical representations.

Bag of words (BoW) a feature extraction method that processes the information to count the frequencies of each word that appear in the text corpora or sentences. The BoW ignores the order of the words in a sentence and considers each word an independent entity. It uses the natural language tool kit NLTK [21] pre-processing techniques like tokenizing sentences, removing stop words and other punctuation symbols from the given text, and building the vocabulary of the known words. The BoW models perform binary counts, i-e: like the given word w is present or absent in each document.

Limitation of BoW models:

- **Lack of context:** The BoW models need to consider the context in which a word appears, which can lead to poor performance on specific tasks such as sentiment analysis.
- **Sparsity:** If most elements are zero, then the BoW will be a sparse matrix. The sparse representations are difficult for model computation because of the high number of input vector weights.
- **Semantic:** The schematic BoW does not preserve the semantics of words, and the order of word appearance in the text is not considered.

More advanced models, such as neural network-based models, like Vanilla transformers and BERT, have been developed to effectively capture context and syntactic and semantic information to overcome these limitations.

WORD EMBEDDING is the solution for the limitations of the high dimensionality of the vectors because it transforms the large sparse vectors into the lower dimensional space. One of the main advantages of word embeddings is that they can capture the meaning and context of words in a way that machine learning models easily understand. In addition, because words with similar purposes will have similar embeddings, these vectors can be used for tasks such as text classification, language translation, and text generation. There are many types of word embedding methods. Popular ones are **Word2vec** and **GloVe**.

“king is to queen as man is to woman”

Consider the order of words in the space, which means the vectors of similar words are placed near each other, as shown in Figure 1.8.

Figure 1.9 shows the first, the temporal word embeddings capture the semantics of each word concerning time, and in the second view, the model



Figure 1.8: Similar words placed near to each other in the space.

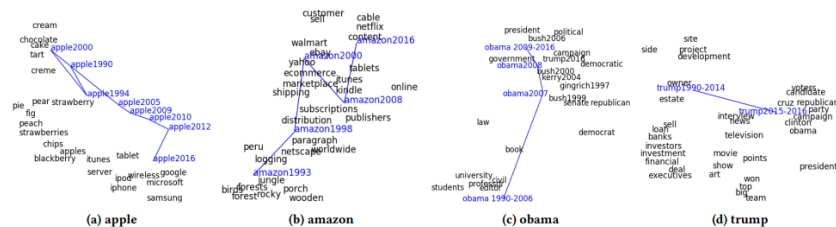


Figure 1.9: Trajectories of brand names and people through time: Apple, Amazon, Obama, and Trump, Image taken from [5].

provides high-alignment quality in the same-meaning of co-related words for different years have geometrically close embeddings [5]. **How can the high dimensional data be transformed into lower dimensional space?** With the help of dimensionality reduction methods like PRINCIPLE COMPONENT ANALYSIS (PCA) that are used to create the embedding of the words. The PCA tries to catch the highly correlated dimensions that can be transformed into a single dimension using the BoW model.

Word2Vec: A Google-invented algorithm used for training word embeddings. It relies on the distributional hypothesis, which means that the semantically similar words can be placed near each other in the space as shown in Figure 1.9. Word2Vec method [5] helps to map semantically similar words to geometrically close embedding vectors. For the distributional hypothesis, it uses the continuous bag of words or the skip grams. A continuous bag of words (CBoW) uses the continuous distribution of the context. It also considers the word's order and the previous history of the words, like collaborative filtering.

Many successful natural language processing tasks have utilized embeddings learned through Word2Vec. The main ideas from the papers [18] for the learning representations of words are the following:

- **Continuous Skip-gram:** The model considers words into a vector one at a time. Each word is scanned within a specific range before and after the current word in the same sentence. The ranges are n-grams, where an n-gram is a contiguous sequence of n items in a linguistic sequence.
- **Continuous Bag-of-Words (CBoW):** This model predicts words based on the average of their vectors. Specifically, the distributed representations of the surrounding words are combined to predict the word in the middle (the current one). For this model, the order of the words is not essential since we take the average.

In practice, Skip-gram has been shown to have good results since it can positively score rare words or phrases, even when the training dataset size is relatively small. Conversely, the computational time to train Continuous Bag-of-Words is way smaller than the skip-gram and has slightly better accuracy for the frequent words.

Global Vector for word representation (GloVe): The GloVe is a log bi-linear regression model [25], used for the unsupervised learning of word representations that outperform then other models on word analogy, word similarity, and named entity recognition tasks. GloVe was developed by Pennington [25], a statistical method that uses global factorization methods like **Latent Semantic Analysis (LSA)**. LSA [25] is a fully automatic mathematical/statistical technique for recognizing latent similarities. In particular, it maps the matrix in a reduced vector space that approximates the original one, focusing on the essence of the data.

1.1.8 Language Models for Open and Closed-domain Question Answering

Open-domain question answering is very challenging because there is no evidence or context given to the question. Still, it retrieves the context using different methods for retrieving text from documents like **TF-IDF or BM25** [26] models. For example, the exam modality of a student by considering the open and closed book setting.

Open-book questions permit using reference materials during the exam, such as textbooks, notes, or other resources. The goal of available book exams is to test the understanding and application of information rather than just memorization. Therefore, open-book exams require more critical thinking and problem-solving skills, as students must locate and effectively use available information.

Closed-book questions do not allow using any reference materials during the exam. Closed-book exams aim to test a student’s recall and memorization of information. These exams require students to understand the subject material strongly, as they must rely solely on their knowledge to answer the questions. Closed book exams emphasize rote memorization and can test a student’s ability to recall specific details and apply information under pressure.

- **Open-domain QA:** where **context is not provided**. The expectation is that the model has *internalised* knowledge within its parameters and should be able to answer the question with additional context.



- **Closed-domain QA:** where **context is provided**. The expectation is that the model has learned to find answers from context.



Figure 1.10: Explains the concept of open and closed-domain question-answering setup.

Closed Domain Question Answering: In general, when implementing a question-answering system, we consider three perspectives for the question-answering system: domain type, system type, and question type. In the domain type, we focus on closed-domain and open-domain type questions. In a closed domain QA (CDQA) [27], the system has some restrictions, meaning the answer must be restricted or taken from specific topics or directions. While on the other hand, for the open domain, there is no restriction on the domain. The system works on every domain and tries to find answers from any direction.

Additionally, CDQA have a predefined set of questions or topics limitations that the system performs reasoning within the predefined boundaries or the available knowledge. For instance, take an example of a sales chatbot. The chatbot system only works and accepts those types of conversion that belong to sales and purchase, not answers to questions considered off-topic to sales and purchase. At the same time, open-domain systems can deal with and process all-natural language questions and then perform reasoning to find the answer from the public domain.

1.2 Knowledge Graphs

A knowledge graph is a data model representing entities and their relationships in a structured manner. It organizes and links data from various sources for easy access and understanding. Knowledge graphs are often used in artificial intelligence and natural language processing applications to improve the accuracy and relevance of search results and provide a more intuitive way to navigate and understand large data sets.

1.2.1 Preliminary Definitions of Graphs

Definition 1: [28] A knowledge graph is a multi-relational graph composed of entities and relations regarded as nodes and different types of edges, respectively.

Definition 2: [29] A knowledge graph mainly describes real-world entities and their inter-relations, organized in a graph. It defines possible classes and relations of entities in a schema and allows for potentially interrelating arbitrary entities with each other. It also covers various topical domains.

These days, graph-based technologies are very popular and are considered very efficient data structures for expressing information in the form of nodes (entities) and edges (relations). Many graphs, such as **Social Networks**, **Biomedical Networks**, **Wikipedia**, etc.

Because of graph efficiency, most famous companies, such as **Google**, **Amazon**, **Facebook**, **Linkedin**, **IBM**, etc., use graphs as a data structure to store and process their information. Moreover, graph data is very flexible and can be easily understood by humans because graphs transform complex data into simpler things like triples that are easy to understand and express.

General structure of the Knowledge Graph: By using NLP techniques, we need to extract and use the important features of graphs, such as:

1. Set of E entities (nodes)
2. Set of K relationships (edges)

1.2.2 Graph Types and their Properties

Simple Graph

Definition 3: A simple graph G [30, 31] is a pair $G = (V, E)$ where.

1. V is a finite set called the vertices of G .
2. $E \subset V \times V$.

Graphs consist of two finite sets, namely, a set of vertexes or a set of nodes and a set of edges. A strong relationship exists between two sets (V, E) , and combining both sets defines the graph's structure. The important feature graph edges that pay attention to or indicate the strength of the connection between the nodes.

Directed Graph

Definition 4: A directed graph (or digraph) [30, 31] has edges with direction, which indicate a one-way relationship. We can define a directed graph as a triple $D = (V, E, \phi)$ where V and E are finite sets and ϕ is a function with domain E and co-domain $V \times V$. We call E the set of edges of the graph D and V the set of vertices of D .

Un-Directed Graph

Definition 5: An un-directed graph [30, 31] instead have edges without direction, which indicates a two-way relationship. In other words, it is a set of vertices (or nodes) connected by edges without direction.

Multi-Graph

We can also define a graph with multiple edges with the same nodes, called Multi-Graphs.

Definition 6: Just like a simple graph, we can define directed and un-directed multi-graphs [30, 31], specifically:

- **Directed Multi-Graph**(multiple edges with directions) G is an ordered pair $G := (V, A)$ where V is a set of nodes and A a multi-set of ordered pairs of vertices called directed edges or arcs.
- **Un-directed Multi-Graph** (multiple edges without directions) G is an ordered pair $G := (V, E)$ where V is a set of nodes and E a multi-set of unordered pairs of vertices, called edges.

1.2.3 Graph Embedding Techniques

Graph Embedding is a technique to transform input features like nodes, edges, and attributes into vector space representation with lower dimensions.

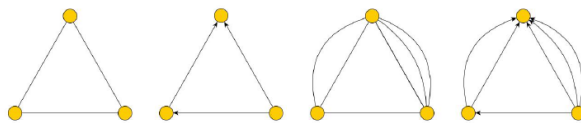


Figure 1.11: Graphical representation of graphs from the left: un-directed graph, directed graph, un-directed multi-graph, and directed multi-graph.

It also preserves the semantics of similar words placed close to each other in the transformed space (Dense Space). Several embedding methods are used in graph pre-processing to convert a graph into a computationally digestible format because graphs are original in discrete nature.

In this section, we will see different embedding levels like node level, sub-graph level, or through with varying strategies of graph walks. These are some of the most popular embedding methods.

Semantics Embeddings and Translational Models

Graph semantic embeddings are low-dimensional vector representations of nodes in a graph. Each node represents an entity, such as a person or an object, and the edges represent relationships between entities. Graph semantic embeddings aim to capture a graph's underlying structure and relationships in a compact and continuous representation.

These embeddings are typically learned using neural network-based methods, such as graph convolutional networks (GCNs) or graph auto-encoders. The learned embeddings are then used for various downstream tasks, such as node classification, link prediction, and recommendation systems. Graph semantic embeddings have proven effective in many applications, including knowledge graphs, social networks, and biological networks, due to their ability to capture the complex relationships in a graph and generalize them to unseen nodes and relationships.

There are several types of semantic graph embeddings, such as:

- **Translational-based methods:** These methods define graph embedding as a translational operation from the node features to a low-dimensional space. Examples include DeepWalk, node2vec, and LINE.

DeepWalk DeepWalk belongs to a family of graph embedding that uses a walk or traverses the graph by moving from one node to another node in the graph. For each traversal step of the graph, the vector representation of nodes is aggregated and placed next to each other in the matrix. After

that, we have an arranged matrix of all node representations fed into the deep learning model for training purposes.

The basic concept of DeepWalk [6] and the Word2vec [32, 18] is quite similar because both follow the same strategy of using the co-occurrence relationship between nodes in the graph is to learn the vector representation of nodes.

The question here is **“How to represent the co-occurrence relationship between nodes.”**

The goal of the deep walk is to estimate the likelihood of observing nodes based on all the previous nodes explored and sampled by a random walk. RandomWalk follows the depth-first search strategy [6] that traverses the nodes repeatedly and tests them. The sampling of currently visited nodes is based on the neighbour nodes, and it repeats and explores the nodes until the length of the visited sequence meets the given condition.

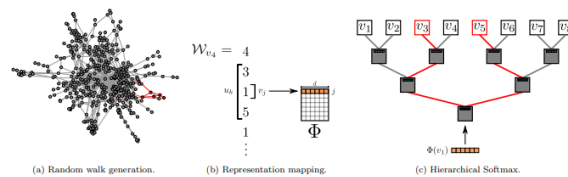


Figure 1.12: How Deepwalk traverses the nodes, Image taken from [6].

- Construct a homogeneous network, starting by sampling Random Walk separately from each node in the network to obtain locally associated training data.
- SkipGram training of sampled data, representing discrete network nodes as direct quantification, maximizing node co-realization, and using Hierarchical Softmax as a classifier for ultra-large-scale classification.

Node2Vec

Node2vec [33] is a graph method that combines DFS (Depth First Search) neighborhoods with BFS (Breadth First Search) neighborhoods. Simply put, it is an extension of the deep walk, which combines DFS and BFS random walk. Node2vec still uses a random walk method to obtain the nearest neighbor sequence of a vertex. The difference is that Node2vec uses a biased random walk.

- **Matrix Factorization-based methods:** These methods factorize the adjacency matrix of a graph into two low-dimensional matrices, one representing the node embeddings. Examples include Graph Factorization and Laplacian Eigenmaps.
- **Neural Network-based methods:** These methods use neural networks to learn graph embeddings. Examples include Graph Convolutional Networks (GCN) and Graph Attention Networks (GAT).
- **Autoencoder-based methods:** These methods use autoencoders to learn node embeddings by reconstructing the graph structure from the node representations. Examples include variational graph autoencoders and graph variational autoencoders.
- **Hybrid methods:** These methods combine to leverage their strengths and overcome limitations. Examples include the combination of matrix factorization and neural network-based methods.

Each type of semantic graph embedding has its advantages and limitations, and the choice of a method depends on the specific problem and the characteristics of the graph.

Translational Models

Translational models model graph relationships by interpreting them as translations in the embedding space. They have received a significant number of attention in the link prediction task [34]. Instead, we have used them to retrieve the embeddings to build up our similarity matrix.

The triplets consist of: (a head entity, relationship, and tail entity). Even though we would not be focusing on the evaluation of the link prediction task. Since we are more interested in the embeddings themselves, we followed the ranking procedure proposed by the literature on benchmark datasets to know also the quality of the embeddings themselves. For each triplet in the test set, the head entity is removed and replaced by each of the entities of the KG dictionary. The evaluation metrics were: Mean of the Predicted Ranks (**MRR**) and HITS@10 (HITS@N indicates the probability that the correct reasoning result appears in the first N results, which is similar to the recall rate of the knowledge reasoning algorithm).

TransE

TransE is a type of embedding technique used in knowledge graph representation learning. TransE stands for Translation Embeddings and is a method for

embedding entities and relationships in a continuous vector space. The main idea behind TransE is to represent relationships as translations between entities. TransE represents each entity and relationship as a vector in a low-dimensional continuous space. The vectors for entities are learned such that the relationships between them are represented as simple vector translations.

For example, suppose the relationship "is a parent of" can be represented as a vector "p". In that case, the relationship "is a parent of" between entities "A" and "B" and can be represented as $A + p = B$. TransE is effective in knowledge graph completion tasks, such as link prediction, where the goal is to predict missing relationships between entities. TransE is also computationally efficient, making it suitable for large-scale knowledge graphs. However, it has some limitations, such as the inability to represent asymmetric relationships and the assumption that relationships are translations, which may only sometimes hold in real-world knowledge graphs.

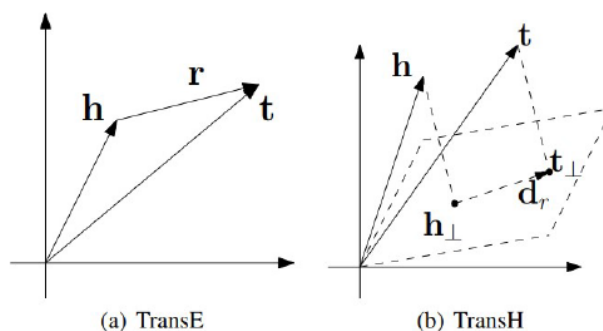


Figure 1.13: Representation of TransE and TransH Models image taken from [7]

TransH

TransH [7] is a type of embedding technique used in knowledge graph representation learning. TransH stands for Translation in Hyperplane and is a method for embedding entities and relationships in a continuous vector space. TransH extends the idea of TransE by considering hyperplanes instead of vector space. The model TransH differs from the previous one since it considers the relationship as a hyperplane where an operation of translation is computed. In this way, the model can handle one-to-many, many-to-one, and many-to-many, which TransE is incapable of while using almost the same complexity.

In TransH, each entity is represented as a vector in a low-dimensional continuous space, and each relationship is represented as a hyperplane. The

vectors for entities are learned such that the relationships between them are represented as translations along the average vector of the hyperplane associated with the relationship. The main advantage of TransH over TransE is its ability to handle relationships that are not translations, such as asymmetric or inverse relationships. TransH can also handle relationships with multiple interpretations by assigning different hyperplanes to different interpretations of the same relationship.

DistMult

DistMult is a type of embedding technique used in knowledge graph representation learning. DistMult stands for Distributed Matrix and is a method for embedding entities and relationships in a continuous vector space. The main idea behind DistMult is to represent relationships as a dot product between the vectors of the entities involved. In DistMult, each entity and relationship is represented as a vector in a low-dimensional continuous space. The vectors for entities are learned such that the relationships between them are represented as dot products. For example, suppose the relationship "is a parent of" is represented as a vector "r". In that case, the relationship "is a parent of" between entities "A" and "B" can be represented as $A \cdot r = B$.

DistMult is effective in knowledge graph completion tasks, such as link prediction, where the goal is to predict missing relationships between entities. DistMult is also computationally efficient, making it suitable for large-scale knowledge graphs. However, it has some limitations, such as the inability to represent anti-symmetric relationships and the assumption that relationships are dot products, which may only sometimes hold in real-world knowledge graphs.

1.2.4 Example of Knowledge Graphs

Google Knowledge Graph

One example of a knowledge graph is Google Knowledge Graph [35]. Google KG is a large-scale knowledge graph developed by Google to enhance its search results with semantically-rich information. The Knowledge Graph aims to understand the relationships between entities, such as people, places, things, and events, and to provide users with a more complete and relevant search experience.

For example, when a user searches for a person "Tom Simpon", the Knowledge Graph might display information about their occupation, gender "Male",

What Google's Knowledge Graph Looks Like

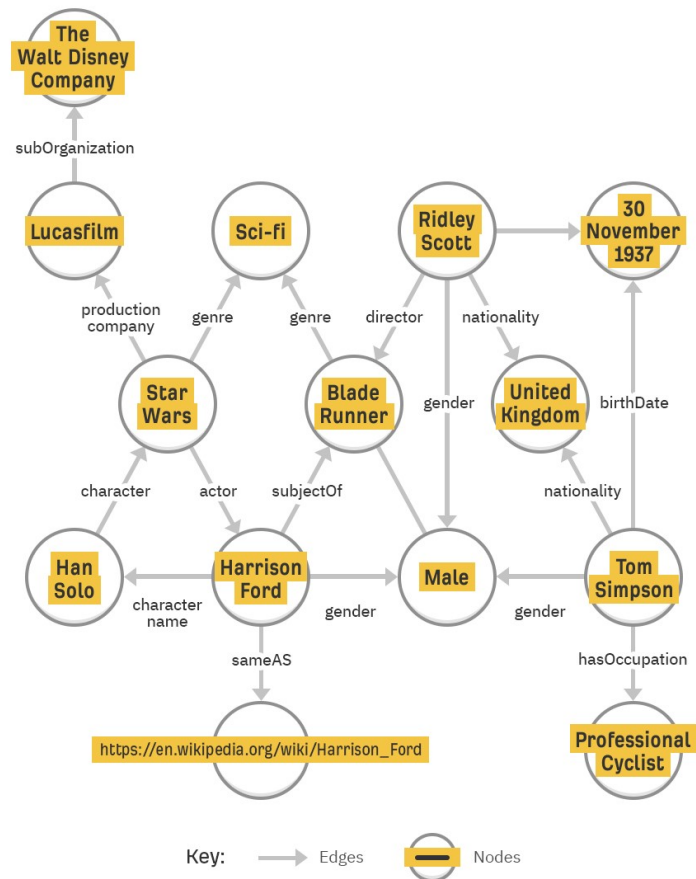


Figure 1.14: Example of Google Knowledge Graph

birth date, notable works, and other relevant details, all in a graphical format as shown in Figure 1.14. Likewise, when a user searches for a place, the Knowledge Graph might display information about its location, history, and notable landmarks, among other details.

DBpedia

Another example of a knowledge graph is **DBpedia [36]**. DBpedia is a large-scale knowledge graph built from the structured data of Wikipedia. It's a community effort to extract and publish information from Wikipedia in a structured and machine-readable format, making it easier for applications to access and use Wikipedia data. In DBpedia [36], entities in Wikipedia articles are represented as nodes in the graph, and the relationships between entities are represented as edges.

For example, an article about a person can be defined as a node. The relationships between that person and other entities, such as their birthplace, education, and employment, can be represented as edges.

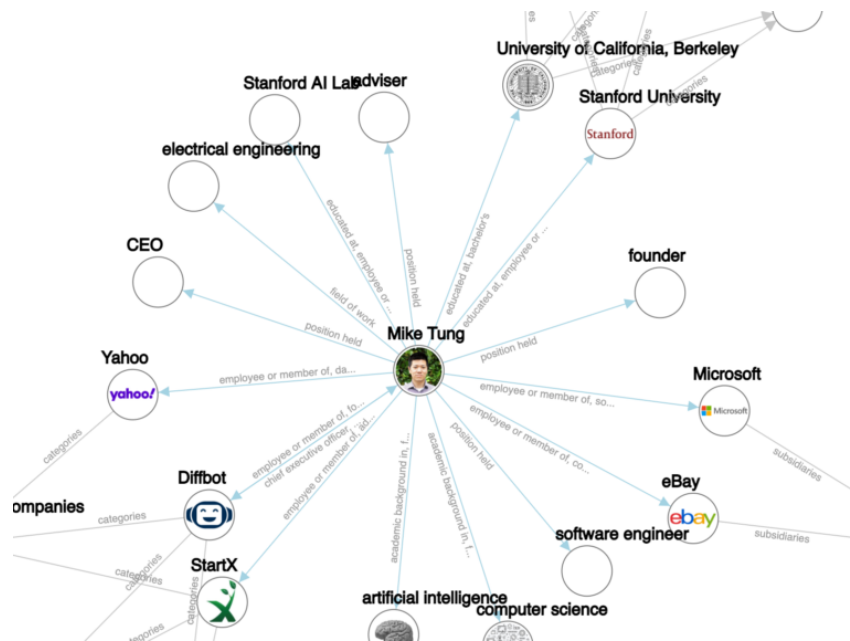


Figure 1.15: Example of world's largest DBpedia Knowledge Graph

DBpedia [36] provides a semantic web interface to the information stored in its knowledge graph, which enables applications to access and query the data using semantic web technologies like **RDF** and **SPARQL**. This makes it possible for developers to build intelligent applications that can understand and reason about the relationships in the data.

Here's an example of how to make a simple query on DBpedia using SPARQL:

```
SELECT ?person ?birthplace
WHERE {
  ?person dbo:birthPlace ?birthplace.
}
LIMIT 10
```

List 1.1: Python example

This query retrieves 10 people and their respective birthplaces from DBpedia. Does the query work by first defining the variables **? person** and **?birthplace**. Then, the WHERE clause specifies the pattern we want to match in the data, in this case, the relationship between a person and their birthplace, represented by **dbo:birthPlace**. Finally, the **LIMIT 10** clause specifies that we only want to retrieve the first 10 results.

1.3 Language Models and Structured Knowledge Graph

Why does the community try to combine the Language Models and Knowledge Graphs for a reasoning process?

Combining Language Models with Structured Knowledge Graphs can create more powerful and versatile AI systems. Knowledge Graphs are structured representations of information that can be used to provide context and background knowledge for Language Models. This can enable Language Models to understand the relationships between different concepts and entities and to make more accurate predictions and recommendations.

For example, people often use search engines and personal assistants to perform daily activities like question-answering or factual information through web searches. For example, we have the input question, **“Who is the Prime Minister of Italy?”**. The information retrieval system requires the background knowledge **“Giorgia Meloni is an Italian politician, she has been serving as the first lady Prime Minister of Italy since 22 October 2022”**. It performs reasoning over it to produce the answer **“Giorgia Meloni”**.

1.3.1 Limitations of Pure Language Models

- **Lack of common-sense understanding:** Language models are trained on large amounts of text data, but this data may need to contain information about common-sense knowledge and reasoning, which can limit

the model's ability to understand the variation of human language and context.

- **Limited ability to handle out-of-vocabulary (OOV) words:** Language models are trained on a fixed vocabulary, so they may be unable to manage words or phrases they have yet to see before.
- **Limited ability to handle multi-turn dialogue:** Language models are typically trained on single-turn inputs and may need help understanding the context and flow of a conversation.
- **Lack of interpretability:** Language models are complex neural networks, and it can be hard to understand why they make predictions, which can be a limitation in specific applications.
- **Limited ability to handle structured data:** Language models are mainly designed for unstructured text data, so they may need help to handle structured data such as tables or graphs.

These limitations can be addressed by combining language models with other NLP techniques, such as knowledge graphs.

1.3.2 Benefits of Combining Language Models and Knowledge Graphs

- **Improved accuracy:** Knowledge Graphs can provide additional context and background information for language models, which can help improve their accuracy and reduce the need for large training data.
- **A better understanding of relationships:** Knowledge Graphs can be used to represent the relationships between different concepts and entities, which can help language models understand the context and meaning of the text.
- **Improved efficiency and scalability:** Structured Knowledge Graphs can enhance the efficiency and scalability of language models LMs by reducing the need for large training data.
- **Better handling of out-of-vocabulary (OOV) words:** Knowledge Graphs can provide additional information about OOV words and phrases, which can help language models understand and generate text more effectively.

- **Better handling of multi-turn dialogue:** Knowledge Graphs can provide additional context and background information that can help language models understand the context and flow of a conversation.
- **Better handling of structured data:** Knowledge Graphs can be used to represent structured data, such as tables and graphs, which can help language models understand and generate text more effectively.
- **Better handling of commonsense reasoning:** Knowledge Graphs can provide additional information about commonsense knowledge and reason, which can help language models understand the nuances of human language and context.
- **Better interpretability:** Knowledge Graphs provide a structured representation of data, which makes it easier to understand why confident predictions are made.

Overall, knowledge Graphs can improve the performance of a wide range of NLP applications, from natural language understanding to the next generation. Structured Knowledge Graphs can also improve language models' efficiency and scalability by reducing the need for significant training data. Overall, combining Language Models and Structured Knowledge Graphs can improve the performance of a wide range of AI applications, from natural language processing to recommendation systems and beyond.

Chapter 2

Related Work

In this chapter, we will focus on the related proposed work to our study, like retrieving information from unstructured data, knowledge-base (Section 2.1), and structured data sources(Section 2.2). Our primary focus is on the Knowledge Graphs (KG): " How to retrieve the relevant portion of the subgraph from a Knowledge Graph? How can we say that the retrieved subgraph is best concerned with its size? How to make or establish a connection with intermediate nodes or missing nodes? " This section will discuss all the challenges, their proposed solutions, and methods.

2.1 Knowledge-based Data (Triplets)

In the Knowledge base, the information is stored as a **triple-like subject, object, and predicate** knowledge base as shown on the right of the given Figure 2.1 using a specific information extraction system. This triple is composed of subject-relation objects.

Our data is available in the form of articles or general documents such as Wikipedia, Commonsense, and many other directories, as shown in Figure 2.1. In addition, NLP techniques, such as information extraction systems, can extract meaningful information from the user input data, encoding the input data into lower dimensions vectors or numerical vectors that are compatible with the training purpose of language models. After getting the standard representation of data, we trained the Language models on general data. Then we used them according to tasks such as Masked Language Modelling or Next Sentence Predictions.

Knowledge sources:

Knowledge Base

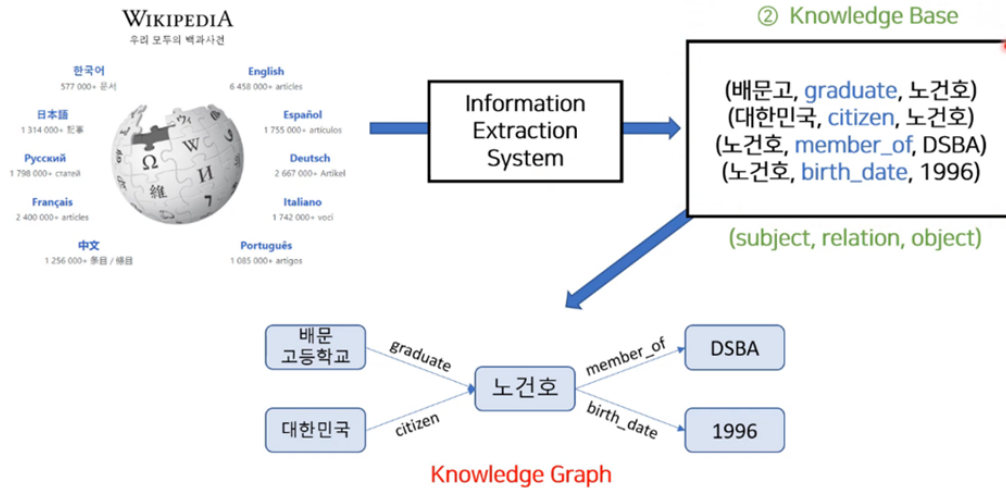


Figure 2.1: Structural Representation of the Knowledge Base and the Knowledge Graph.

1. Language model (LM) (Section 1.1.5): All the available knowledge can be encoded into pre-trained LM's like GOOGLE BERT [3], ROBERTA [22], BIO-LINK BERT [37] etc.
2. Knowledge Graphs (KGs) (Section 1.2): For structured Knowledge, the most popular general-domain KGs are CONCEPT-NET[38], and Freebase [39] that represent knowledge in the form of entities and relations (head-predicate-tail triples).
3. Knowledge base triples (Section 2.1) (Head, Relation, Tail).

In the knowledge base, how to represent the knowledge in the form of triples is shown in Figure 2.1. For example, in the knowledge base, **I am a member of the DSBA**, and I store the information that I belong to the **DSBA laboratory** in the form of a triple and graphs, as shown in the image. The KB and the KG graph have different representations of the same information. So the difference between both sources is only the representation of the knowledge. For example, **Faisal** is a member of the **DSBA** laboratory. So **Faisal and DSBA** is the subject and objects in the graph called nodes, and the edge **Member of** between subject-object is called the relation.

2.2 Reasoning with Language Models and Knowledge Graphs

Structured KG is more popular than KG, and the LM does not capture the semantic meaning of the same context with billions of parameters. LM cannot capture the semantic essence of similar sentences. A structured way to store information is called knowledge graphs. Knowledge Graphs KG is the backbone of multiple applications, such as information retrieval systems like Search Engines, Chabot's, and many Question-and-Answering applications. It is also known as the semantic network that represents the information in the form of real-world entities like the Subject, Object, and Predicate.

The basic structure of the knowledge graph is based on the three modules the set of nodes, the group of edges, and the relationship between them—the triples where we have the **head, predicate, and tail**. For example, **nodes (heads and tails)** can be any real work entities like **Person, Place, Animals, or things like IBM, Faisal, Lion, Italy**, etc., and predicates or edges between them showing the relationship between the given entities. **For example, Faisal is a student, so the IS-A relationship expresses the entities of Faisal and the students.**

To better understand the subgraph retrieval from existing Knowledge Graphs, we have similar existing proposed solutions, but they have some constraints on the retrieval phase. While studying all of these papers in the upcoming section can clarify our study and demonstrate the difference between their proposed solution and our suggested Subgraph Retrieval method.

2.2.1 QA-GNN

Generally, the information can be implicitly encoded in the large language models (LMs) [17] pre-trained on unstructured text, as explained in section 02. Or explicitly can be described in the structured KGs such as **Freebase** [39] and **conceptNet** [38]. The paper presents two challenges during combining LMs and KGs to perform joint reasoning on the given question-answer input.

Combining LMs and KGs for joint reasoning presents two challenges:

- It's hard to identify the relevant knowledge concerning the given QA context from large KGs.
- It's unable to perform joint reasoning over QA context (LM) and KG.

QA-GNN [14] is one of the first works that address the above challenges through two key innovative methods:

- **Relevance scoring:** since the KG subgraph consists of all few-hop neighbours of the topic entities, some entity nodes are more relevant than others concerning the given QA context. They propose KG node relevance scoring: they score each entity on the KG subgraph by concatenating the entity with the QA context and calculating the likelihood using a pre-trained LM. This presents a general framework to weigh information on the KG;
- **Joint reasoning:** they design a joint graph representation of the QA context and KG, where they explicitly view the QA context as an additional node (QA context node) and connect it to the topic entities in the KG subgraph. This joint graph, which they termed the working graph, unifies the two modalities into one graph. They then augment the feature of each node with the relevance score and design a new attention-based GNN module for reasoning.

Dataset: The Model evaluation on QA benchmarks in the Commonsense (CommonsenseQA, OpenBookQA) and biomedical (MedQA-USMLE) domains.

Approach

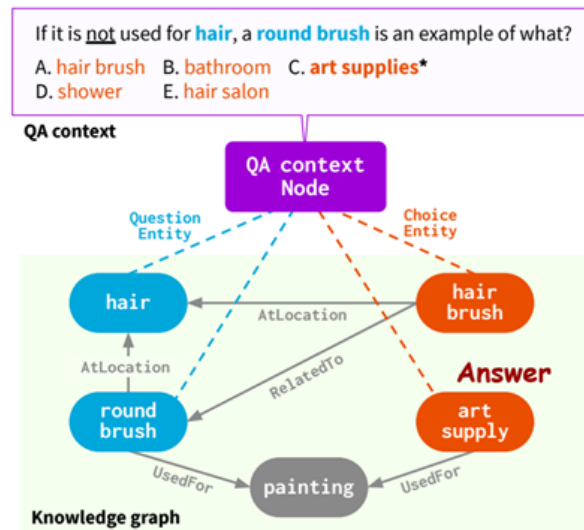


Figure 2.2: Overview of their approach. Given a QA context (z), they connect it with the retrieved KG to form a joint graph, compute the relevance of each KG node conditioned, and perform reasoning on the working graph

As shown in the figure above, given a question q and an answer choice a , they concatenate them to get the QA-context $[\mathbf{q}; \mathbf{a}]$. To reason over a given QA context using knowledge from both the LM and the KG, QA-GNN works as follows:

- They use LMs to obtain a representation for the QA context and retrieve the subgraph from the KG.
- Then they introduce a QA context node \mathbf{z} that represents the QA context and connects \mathbf{z} to the topic entities so that they have a joint graph over the two sources of knowledge, which they term the working graph. To adaptively capture the relationship between the QA context node and each of the other nodes, they calculate a relevance score for each pair using the LM and use this score as an additional feature for each node.
- Finally, they make the final prediction using the LM representation, QA context node representation, and a pooled working graph representation.

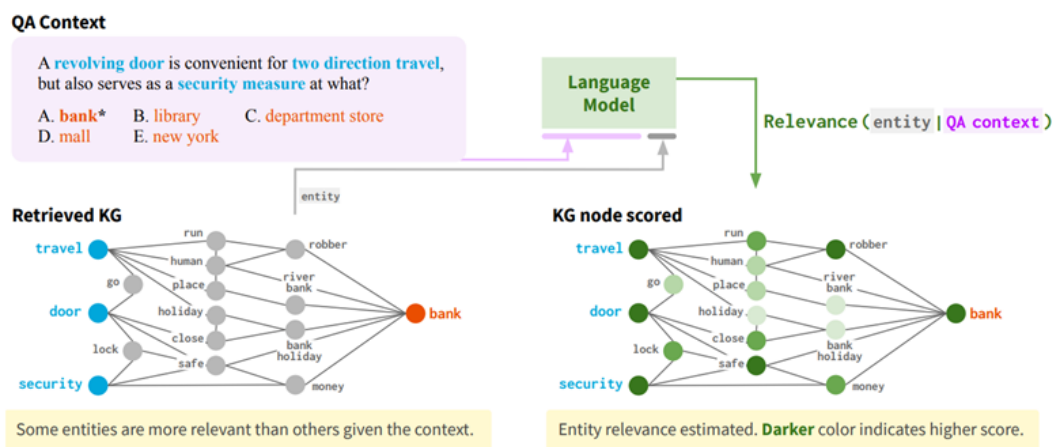


Figure 2.3: Relevance scoring of the retrieved KG: they use a pre-trained LM to calculate the relevance of each KG entity node conditioned on the QA context.

They compared with existing LM+KG methods, which share the same high-level framework as their but use different modules to reason on the KG in place of QA-GNN. The critical differences between QA-GNN and these are that they do not perform relevance scoring or joint updates with the QA.

The image below shows the result of QA-GNN on MedQA-USMLE. QA-GNN outperformed fine-tuned LMs (e.g., SapBERT [40]).

| Methods | Test |
|----------------------------------|-------------|
| BERT-base (Devlin et al., 2019) | 34.3 |
| BioBERT-base (Lee et al., 2020) | 34.1 |
| RoBERTa-large (Liu et al., 2019) | 35.0 |
| BioBERT-large (Lee et al., 2020) | 36.7 |
| SapBERT (Liu et al., 2020a) | 37.2 |
| SapBERT + QA-GNN (Ours) | 38.0 |

Figure 2.4: Test accuracy on MedQA-USMLE

To retrieve the relevant nodes or portion of the sub-graph from the large Knowledge graph is quite challenging in this paper as well in the other relevant documents like GreaseLM and Dragons have the same issue.

- First, in structured Knowledge Graphs, the size of the retrieved graph is very challenging because if the retrieved graph is too small so there are chances of missing the answer because of the small size, and if it is retrieved large, that can cause to introduce the noise that may effect on the reasoning phase. Earlier methods use different strategies like the retrieved the top-K entities from the knowledge base triple store (head, relation, tail). This approach can also retrieve some off-topic entities that do not belong to the question, as mentioned in the QA-GNN.
- Second, the issue about the weak internal intermediate node or entities, which means making or establishing a connection with the intermediate node. Suppose we look into space when we want to explore the space and start from topic entities and explore other paths until we find the desired answer in the space; if there is no answer in the space, which direction do we follow to go through in between them answer and the question. So, the fact that there is no correct answer for the intermediate process is called the state of Weak Internal Media Supervision, which is also faced in the existing studies.

This challenge can be seen in the dataset, only the question and its correct answer are given, and there is no answer to the intermediate nodes of which multi-hop process to find the correct answer. (Intermediate node answers missing). Therefore, the extracted KG subgraph ignores some intermediate concept (entity) nodes and edges. In such cases, the subgraph does not contain a complete chain of reasoning.

- Third, an Incomplete subgraph is retrieved without complete information about the given entities and relations or the neighbor nodes, etc.
- Previous work (DRGN, CBR) employs very different technical solutions for developing the retrieval and reasoning models, neglecting their relatedness in task essence.

The recent research QA-GNN, Grease LM, and the Dragon model is the first work to deal with the negative questions. QA-GNN improves the reasoning under negation by adding the QA global node to the graph. However, the challenge still exists. These are significant challenges mentioned above. However, these challenges are the issues that the various methodologies currently being researched are trying to solve. This study demonstrates how to solve these issues in a new working framework.

2.2.2 GreaseLM: Graph Reasoning Enhanced Language Models For Question Answering

Answering the complex questions about textual representation requires more reasoning over the stated context and the world knowledge that motivates it. The **GreaseLM** [8] presents a new model architecture that uses the jointly encoded representations from pre-trained LMs and graphs neural networks over multiple layers of modality interaction operations. It shows how two sources of knowledge work together to perform QA, as shown in Figure 2.5. The primary **GreaseLM focuses on the reasoning phase**, which deals with the negations and hiding conditions like those tested in QA-GNN.

Limitations of Existing Methods:

- After pretraining these Language models on a large-scale collection of general text corpora, these language models learn to encode overall knowledge about the world, which implicitly they can leverage when fine-tuned on a domain-specific downstream QA task. However, these fine-tuned models struggle when the given examples are distributionally different from examples seen during the fine-tuning phase. The behavior of the fine-tuned language models often relies on simple patterns to present the shortcut answer rather than robust or the concerned one, while structured reasoning that effectively combines the explicit information provided by the context and implicit external knowledge for reasoning and inferring the correct answer instead of the presenting the shortcut answer.

- Existing methods typically combine two modalities (LM and KG) in a shallow and non-interactive mode, encoding both sources of knowledge separately and combining them at the output phase for a prediction or using one to augment the input of the other.
- The existing method fails to deal with multiple constraints, such as negation and hedges conditions that require practical reasoning over both language context and KG.
- The subgraph retrievals are similar to the work of QA-GNN and Dragons, which is inefficient concerning the size and the retrieved off-topic entities.

Dataset: GreaseLM tested on the famous benchmark dataset for common-sense and general question-answering reasoning over datasets like COMMONSENSEQA and OPENBOOKQA. For reasoning over medical domain type question answering (i.e., MEDQA-USMLE). The domains demonstrate that GreaseLM can more reliably answer questions requiring reasoning over situational constraints and structured Knowledge.

Approach

For dealing with the complex QA and the limitation of the fine-tuning of language models, negotiating with and resolving the multiple constraints such as negation or conditional constraints. GreaseLM [8] presents a **multi-layer architecture** graph neural network. For each layer in the following network, the representation of vectors is updated after each iteration to perform efficient reasoning on the retrieved subgraph to infer the correct answer. The image can explain the methodology used in GreaseLM.

GreaseLM Architecture Description:

In phase one, GREASELM takes input representations of both modalities, expressive large language models, and structured KGs. The KG retrieval that retrieves the subgraph from the KG given the QA context is similarly mentioned in the QAGNN [14] because the subgraph retrievals phase is similar in this paper as well in the Deep Bidirectional Language-Knowledge Graph [9]. For the second source of the input, pass the input question through the multiple language model encoder layers called uni-model encoder and takes its representations and use it in the other phase of the framework.

In the second phase of the GREASELM [8], the cross-modal fuser takes both sources of the inputs and mixes the information from KG and language representations for bidirectional integration's. GREASELM has three components; the first transformer LM encoder block continuously encodes the language context, and the second GNN layer reasons over the KG entities and relations. The third

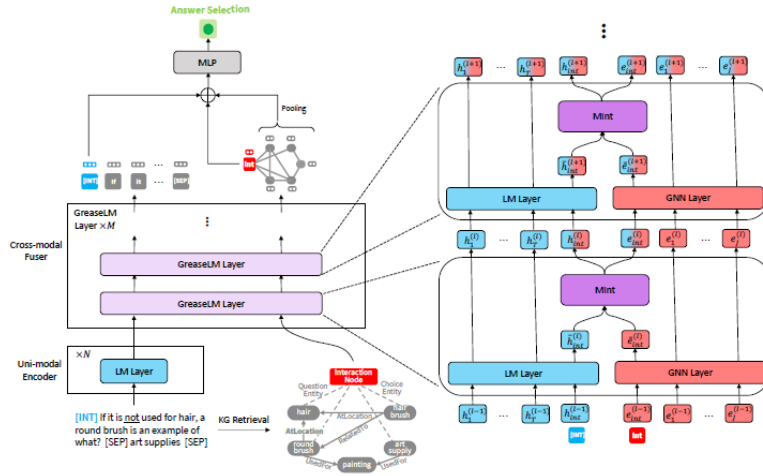


Figure 2.5: General architecture of **GreaseLM** taken from [8]

one, called the mint modality operator, takes the uni-model representation of interaction tokens and nodes and exchanges the information through them.

Two stacked components of GreaseLM:

1. Set of uni-modal LM layers, which learn an initial representation of the given input tokens from both sources of input LMs and KGs.
2. Set of upper cross-modal GREASELM layers, which is learned to represent the language sequence jointly and linked knowledge graph, allowing the textual representations formed from the underlying LMs layers and a graph representation of the KG to mix.

In the GREASELM [8], it focuses on the reasoning phase with multi-layer attention model architecture that performs better reasoning as compared to the existing QA-GNN [14] in some of the cases like dealing with the negation, hedging, or flip of the entities, etc. it is considered as the extended version of QA-GNN, especially in the reasoning phases. GREASELM also uses the same subgraph retrievals phase as in the QA-GNN; at the end, we will replace and test the subgraph retrieval phase with our proposed solution.

Dealing with negation and Flipped conditions:

For example, the model GREASELM correctly predicts that the answer to the given question is “**airplane**” while on the other hand, QA-GNN predicts the incorrect one as “**motor vehicle**”. For both models, GREASELM [8] and QA-GNN [14], they perform the Best First Search (BFS) on the retrieved KG subgraph G_s to trace high attention weights from the interaction node (purple), as shown in the figure 2.5.

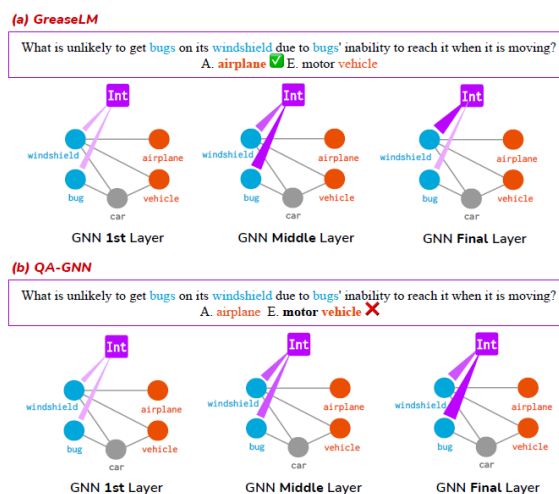


Figure 2.6: Qualitative analysis of GREASELM’s graph attention weight changes across multiple messages passing layers compared with QA-GNN. GREASELM demonstrates attention change patterns that more closely resemble the expected change in focus on the “bug” entity.[8]

In each layer in the network, the representation of each modality is updated, and the representations of the interaction token and node are pulled, concatenated, and passed through a modality interaction (**Mint**) unit to mix their representations. In the following layers, the mixed information from the interaction elements combines with their respective modalities, allowing knowledge from the KG to affect the representations of individual tokens and context from language to affect fine-grained entity knowledge representations in the GNN.

The new model GreaseLM architecture enables deep fusion and exchanges the information from both sources of knowledge in the multiple layers of its architecture that enhance the reasoning process and handle the condition like negation and flipping as shown in Figure 2.6.

2.2.3 Dragon: Deep Bi-directional Language Knowledge Graph Pre-training

A self-supervised method called DRAGON (Deep Bidirectional Language-Knowledge Graph Pre-training) [9] pre-trains a deeply joint language-knowledge foundation model from both sources like textual and KG at any scale. DRAGON model takes the input pairs of text segments, and the relevant KG sub-graph (the **KG sub-graph retrievals phase is the same as used in QA-GNN**

| Methods | Acc. (%) |
|---------------------------------------|-------------|
| Baselines (Jin et al., 2021) | |
| CHANCE | 25.0 |
| PMI | 31.1 |
| IR-ES | 35.5 |
| IR-CUSTOM | 36.1 |
| CLINICALBERT-BASE | 32.4 |
| BIOBERTA-BASE | 36.1 |
| BIOBERT-BASE | 34.1 |
| BIOBERT-LARGE | 36.7 |
| Baselines (Our implementation) | |
| SapBERT-Base (w/o KG) | 37.2 |
| QA-GNN | 38.0 |
| GREASELM (Ours) | 38.5 |

Figure 2.7: Performance on MedQA-USMLE take from [8], demonstrate that GreaseLM outperforms state-of-the-art fine-tuned LMs (e.g., SapBERT) and a QA-GNN augmentation of SapBERT.

[14], and GREASELM [8] as input and bi-directionally joins information from both modalities of inputs. DRAGON pre-train on two self-supervised reasoning tasks: **Masked Language modeling** and the **Knowledge Graph link predictions**.

Approach

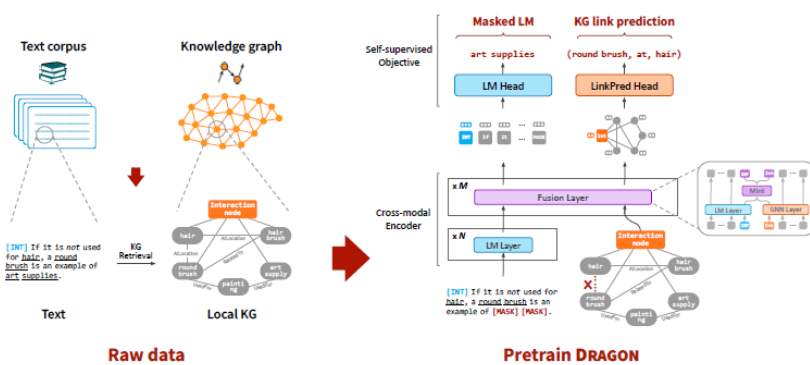


Figure 2.8: General architecture of DRAGON taken from [9]

Language models (LMs) are pre-trained on large amounts of text data, such as **BERT** [3], and GPTs, and have shown strong performance on many natural language processing (NLP) tasks. However, effectively combining text and KGs for pre-training is an open problem and presents challenges.

The core component of DRAGON:

1. Deeply bi-directional model for the two modalities (LM and KG) to interact.

2. Self-supervised task to learn joint reasoning over text and KG.

Description of Dragon:

On the left of the image, the given input, raw data of a text corpus, and an extensive knowledge graph construct the aligned **(text, local KG)** pairs by sampling a text segment from the text corpus and extracting a relevant subgraph from the large KG. As in structured knowledge, KG can ground the text, and the text can provide the KG with a rich context for reasoning. The Dragon Models aims to pre-train a language-knowledge model on the right side of the image for a common reason on the text-KG pairs (DRAGON).

To model the interactions over text and KG, DRAGON [9] and GreaseLM [8] work similarly to using a cross-modal encoder that bi-directionally exchanges information between them to produce fused text token and KG node representations.

To pre-train DRAGON jointly on text and KG, we unify two self-supervised reasoning tasks:

1. Masked Language Modeling (MLM).
2. Link prediction or Knowledge graph link prediction.

Knowledge graph link prediction [34] holds out some edges from the input KG and then predicts them. This joint objective encourages text and KG to mutually inform each other, facilitating the model to learn joint reasoning over text and KG. **Masked Language Modelling MLM** took some tokens as input and masked some of them, and tried to predict those masked tokens again from the sequence using a language model encoder like BERT [3].

The most crucial difference between the two approaches is: Remaining setting for all models is the same as used in **GraseLM and QA-GNN**. As we use the same encoder architecture as GreaseLM for DRAGON, the only difference from GreaseLM is that DRAGON performs self-supervised pre-training while GreaseLM does not.

DRAGON Evaluation

Figure (2.9) shows the accuracy of downstream commonsense reasoning tasks. DRAGON consistently beats the existing LM (RoBERTa) and KG-augmented QA models (QAGNN, GreaseLM) on all tasks. The gain is especially significant on tasks with small training data (OBQA, Riddle, ARC) and tasks requiring complex reasoning (CosmosQA, HellaSwag).

Downstream evaluation tasks: they fine-tune and evaluate DRAGON on three popular biomedical NLP and reasoning benchmarks: MedQA-USMLE

| | CSQA | OBQA | Riddle | ARC | CosmosQA | HellaSwag | PIQA | SIQA | aNLI |
|----------------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|
| RoBERTa [18] | 68.7 | 64.9 | 60.7 | 43.0 | 80.5 | 82.3 | 79.4 | 75.9 | 82.7 |
| QAGNN [8] | 73.4 | 67.8 | 67.0 | 44.4 | 80.7 | 82.6 | 79.6 | 75.7 | 83.0 |
| GreaseLM [9] | 74.2 | 66.9 | 67.2 | 44.7 | 80.6 | 82.8 | 79.6 | 75.5 | 83.3 |
| DRAGON (Ours) | 76.0 | 72.0 | 71.3 | 48.6 | 82.3 | 85.2 | 81.1 | 76.8 | 84.0 |

Figure 2.9: Performance on the 9 downstream commonsense reasoning tasks. [9]

| Method | MedQA | PubMedQA | BioASQ |
|----------------------|-------------|-------------|-------------|
| BioBERT [74] | 36.7 | 60.2 | 84.1 |
| PubmedBERT [75] | 38.1 | 55.8 | 87.5 |
| BioLinkBERT [19] | 44.6 | 72.2 | 94.8 |
| + QAGNN | 45.0 | 72.1 | 95.0 |
| + GreaseLM | 45.1 | 72.4 | 94.9 |
| DRAGON (Ours) | 47.5 | 73.4 | 96.4 |

Figure 2.10: Accuracy on biomedical NLP tasks. DRAGON outperforms all previous biomedical LMs.

(MedQA), PubMedQA, and BioASQ provide details on these tasks and data splits as shown in Figure 2.10.

Figure 2.10 summarizes the model performance on the downstream tasks. Across tasks, DRAGON outperforms all the existing biomedical LMs and KG-augmented QA models, e.g., +3% absolute accuracy boost over BioLinkBERT and +2% over GreaseLM on MedQA, achieving new state-of-the-art performance on these tasks. This result suggests the significant efficacy of KG-augmented pretraining for improving biomedical reasoning tasks. Combined with the results in the general commonsense domain, our experiments also suggest the domain-generalizability of DRAGON, serving as an effective pretraining method across domains with different combinations of text, KGs, and seed LMs.

2.2.4 UNIKGQA: Unified Retrieval and Reasoning For Solving Multi-Hop Question Answering Over Knowledge Graph

Existing Approach:

Existing work (QA-GNN, GREASELM, DRAGON, CBR AND DRGN) usually adopt a two-stage approach for multi-hop KGQA.

- It first retrieves a relatively small subgraph related to the question entities called the topic entities.
- Then, the reasoning on the subgraph is performed to accurately find the answer entities from the retrieved subgraph.

Although these two stages are highly related, previous work employs very different technical solutions for developing the retrieval and reasoning models, neglecting their relatedness in task essence.

Limitation of UNIKGQA: To cope with the vast search space, Can we design a unified model architecture with the same essence for both stages to derive a better performance? To develop a unified model architecture for multi-hop KGQA, a significant merit is that we can tightly relate the two stages and enhance the sharing of relevant information.

However, there are two major issues to developing a unified model architecture for multi-hop KGQA:

1. How to manage very different scales of search space for the two stages? Problem Faced with the vast search space in large-scale KGs, existing work typically adopts a retrieval-then-reasoning approach, which can achieve a good trade-off between accuracy and efficiency for multi-hop KGQA. Generally, the retrieval stage aims to extract relevant triples from the large-scale KG to compose a relatively more minor question-relevant subgraph. In contrast, the reasoning stage focuses on accurately finding the answer entities from the retrieved subgraph.
2. How to effectively share relevant information across the two stages in the network? The purposes of the two stages are different; both sets need to evaluate the semantic relevance of a candidate entity concerning the question (for removal or re-ranking), which can be considered a semantic matching problem in essence. For measuring entity relevance, relation-based features, either direct relations or cross-relation paths, be particularly useful for building semantic matching models.

Although the two stages are highly related, previous studies usually treat them separately in model learning: only the retrieved triples are passed from the retrieval stage to the reasoning stage, while the helpful rest signal for semantic matching has been neglected in the pipeline framework.

Approach

Nowadays, Multi-hop KGQA has become a vital research subject that aims to identify the answer entities of natural language questions from KGs such as Freebase, DBpedia etc. Therefore, recent studies mainly focus on multi-hop KGQA. In this more complex scenario, sophisticated multi-hop reasoning over edges (or relations) is required to infer the correct answer on the KG. In Figure 2.11, For example, given the question “**Who is the wife of the nominee for The Jeff Probst Show**” the task goal is to find a reasoning path from the topic entity “**The Jeff Probst Show**” to the answer entities “**Shelley Wright**” and “**Lisa Ann Russell**”.

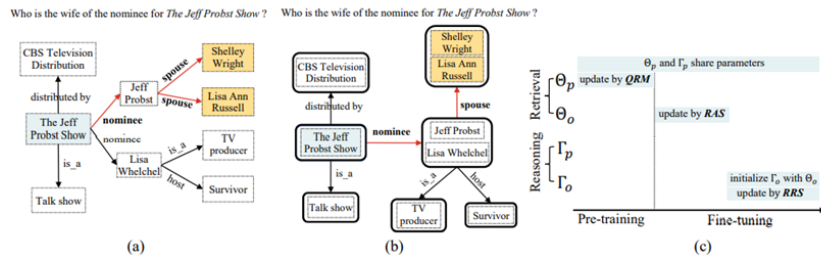


Figure 2.11: Illustrative examples for our work: (a) an example of multi-hop KGQA; (b) an example of abstract subgraph; and (c) the overall learning procedure of our UniKGQA model [10].

As shown in Figure (2.11 a), given the question, it is vital to identify the semantically matched relations and the composed relation path in the KG (e.g., “nominee \rightarrow spouse”) for finding the correct answer entities since the two stages cope with different scales of search space on KGs (e.g., millions vs thousands).

For the first issue, instead of letting the same model architecture directly fit very different data distributions, we propose a new subgraph form to reduce the node scale at the retrieval stage, namely an abstract subgraph that is composed by merging the nodes with same relations from the KG (see Figure (2.11 b)).

For the second issue, based on the same model architecture, we design a practical learning approach for the two stages so that we can share the same pre-trained parameters and further use the learned retrieval model to initialize the reasoning model (see Figure (2.11 c)).

Unified Model Architecture:

We consider a general input form for both retrieval and reasoning and develop the base architecture by integrating two major modules:

- The **Semantic Matching (SM)** module employs a PLM to perform the semantic matching between questions and relations.

- The **Matching Information Propagation (MIP)** module that propagates the semantic matching information on KGs. We present the overview of the model architecture in Figure 2.12. Next, we describe the three parts in detail.

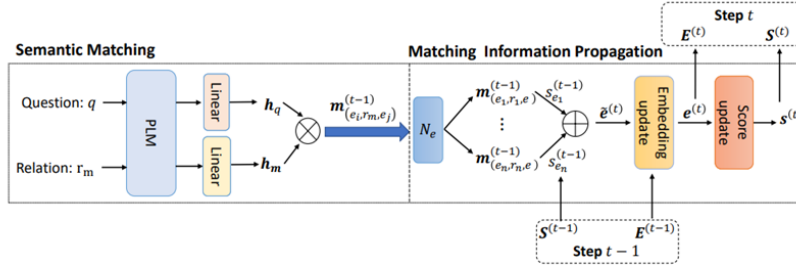


Figure 2.12: The overview of the unified model architecture of UniKGQA, consisting of two modules, i.e., semantic matching and matching information propagation ([10]).

Semantic Matching (SM): The SM module aims to produce the semantic matching features between the question q and a triple h, r, t from the given subgraph G_q . Considering the excellent modeling capacity of the PLMs, we leverage the PLM to produce text encoding as the representations of question q and relation r . Specifically, we first utilize the PLM to encode the texts of q and r , and employ the output representation of the $[CLS]$ token as their representations.

Matching Information Propagation (MIP): Based on the generated semantic matching features, the MIP module firstly aggregates them to update the entity representation and then utilizes it to obtain the entity match score. To initialize the match scores, given a question q and a subgraph G_q , for each entity $e_i \in G_q$.

Dataset: To evaluate this approach, the Model evaluation on QA benchmarks On the simple dataset, MetaQA, Commonsense (CommonsenseQA, OpenBookQA), and biomedical (MedQA-USMLE) domains.

UNIKGQA [10] is the latest research work focused on developing a system for multi-hop question answering over knowledge graphs. The system aims to integrate both retrieval and reasoning capabilities to answer complex questions requiring information from multiple sources accurately. The goal is to provide a unified approach to question answering over knowledge graphs by combining various techniques and models to solve these complex questions efficiently. Multi-hop KGQA aims to find the answer entities from the large-scale Knowledge

| Models | MetaQA-1 | MetaQA-2 | MetaQA-3 | WebQSP | | CWQ | |
|-------------|-------------|------------|------------|-------------|-------------|-------------|-------------|
| | Hits@1 | Hits@1 | Hits@1 | Hits@1 | F1 | Hits@1 | F1 |
| KV-Mem | 96.2 | 82.7 | 48.9 | 46.7 | 34.5 | 18.4 | 15.7 |
| GraftNet | 97.0 | 94.8 | 77.7 | 66.4 | 60.4 | 36.8 | 32.7 |
| PullNet | 97.0 | 99.9 | 91.4 | 68.1 | - | 45.9 | - |
| EmbedKGQA | 97.5 | 98.8 | 94.8 | 66.6 | - | - | - |
| NSM | 97.1 | 99.9 | 98.9 | 68.7 | 62.8 | 47.6 | 42.4 |
| TransferNet | 97.5 | 100 | 100 | 71.4 | - | 48.6 | - |
| SR+NSM | - | - | - | 68.9 | 64.1 | 50.2 | 47.1 |
| SR+NSM+E2E | - | - | - | 69.5 | 64.1 | 49.3 | 46.3 |
| UniKGQA | 97.5 | 99.0 | 99.1 | 75.1 | 70.2 | 50.7 | 48.0 |
| w QU | 97.6 | 99.9 | 99.5 | 77.0 | 71.0 | 50.9 | 49.4 |
| w QU,RU | 98.0 | 99.9 | 99.9 | 77.2 | 72.2 | 51.2 | 49.0 |

Figure 2.13: Performance comparison of different methods for KGQA (Hits@1 and F1 in percent). We copy the results for TransferNet and others. Bold and underlined fonts denote the best, and the second-best methods [10].

Graph entities. The answer entities are multiple hops far from the topic entities mentioned in each question.

2.2.5 CODER: Knowledge Infused Cross-Lingual Medical Term Embedding For Term Normalization

Existing Challenges

Analyzing the free text in Electronic medical records EMRs is challenging because the same medical concept can appear in various nonstandard names, abbreviations, and misspellings, which all need to be normalized to their corresponding standard terms or their concept IDs of existing terminology systems, such as the Unified Medical Language System (UMLS) [41] or SNOMED Clinical Terms (SNOMED-CT)[42]. Normalization is also essential for structured EMR data. For example, medical institutes use different coding systems for laboratory examinations, procedures, and medications. Thus, normalization can facilitate mapping the database fields to the same standard for joint analysis when conducting multi-institutional studies.

Existing medical embeddings can be classified into word, concept, and contextual. Word and concept embeddings have good base performances on evaluating similarity but face the out-of-vocabulary (OOV) problem and cannot handle misspellings commonly present in clinical text. Pretrained Language Models PLM-based contextual embeddings can relieve the OOV problems using sub-word tokenization. Still, they perform less than word and concept embeddings on evaluating similarity if not fine-tuned.

Normalizing medical terms to these systems remains challenging for the classification approach due to the massive number of classes and the inflexibility of adding new classes. Another thing is to rank candidate concepts similar to the term. It differs from the previous approach in that it is trained as a

binary classifier, where terms and their related concepts' names (target terms) form positive samples, and terms and non-corresponding concepts' names form negative samples. The scalar output from the classifier is then used as a measure of similarity to rank the candidate concepts for normalization. Similar to the classification approach, these models are trained on labeled datasets with limited target concepts.

Approach

The CODER [11] is specially designed for medical term normalization by providing relative vector representations for different terms representing the same or similar medical concepts with cross-lingual support. The training of CODER is via contrastive learning on a medical knowledge graph (KG) named the Unified Medical Language System, where similarities are calculated utilizing both terms and relation triplets from KG. **Contrastive learning** is a deep learning technique that involves learning to map similar examples to nearby points in an embedding space while mapping different examples to distant points. By training a model to perform this mapping, it can learn a powerful representation of the data that captures underlying relationships and similarities.

Contrastive learning is a widespread technique in computer vision (CV) as a solution to learning representations in many classes. In each training step, instead of learning to map samples to their labels, contrastive learning aims to learn sample representations to distinguish samples of the same label (positive pairs) from those of different labels (negative pairs), as shown in Figure. For term normalization, terms of the same and different concepts naturally serve as positive and negative pairs in contrastive learning, explored by SapBERT [43] using the UMLS [41] terminology.

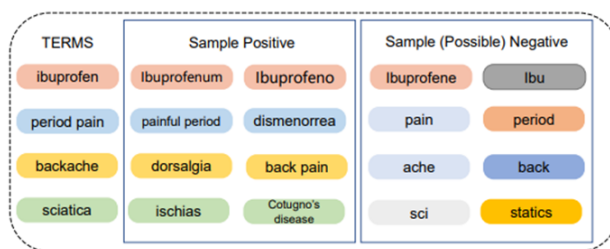


Figure 2.14: Positive and Negative labels pairs, image taken from [11].

However, as a KG, the UMLS contains more knowledge than just terms, and knowledge can serve as helpful information for normalization. For example,

when normalizing “poisoned by eating pufferfish,” we may want “food poisoning” to rank among the top because pufferfish is a kind of food. For another example, it would be more meaningful to have the embedding of “rheumatoid arthritis” closer to “osteoarthritis” than “rheumatoid pleuritis” because both “rheumatoid arthritis” and “osteoarthritis” are subtypes of arthritis, which may affect the same body parts and share treatments. The reasoning in these examples illustrates that relational knowledge is essential to achieve meaningful medical embeddings. Therefore, we propose dual contrastive learning on both terms and relation triplets of KGs. First, the term-relation-term similarity is measured between a term-relation $(, r)$ and a term (t) .

CODER Model Architecture

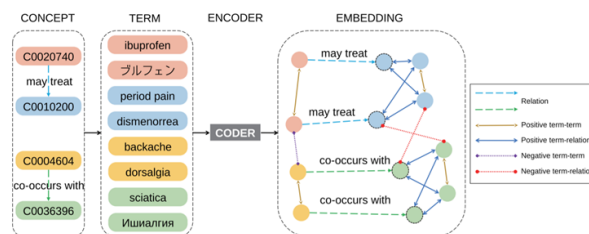


Figure 2.15: The overview of CODER. CODER encodes terms potentially in different languages into the embedding space. Term-term similarities and term-relation-term similarities are calculated to train CODER.

To support cross-lingual term normalization, we use mBERT [44], which encodes texts in different languages in one unified space. Previous methods are usually translation-based and rely on biomedical parallel corpora. The UMLS naturally contains massive cross-lingual medical synonyms which have yet to be fully utilized. The contributions of this paper include the following:

- They propose a KG contrastive learning model for term normalization that uses both synonyms and relations from the UMLS to direct the generation of medical term embeddings. The learning strategy can also be applied to other KGs.
- We perform evaluations on CODER against notable existing medical embeddings. CODER achieves state-of-the-art results in zero-shot term normalization, medical concept similarity measure, and concept relation classification tasks. CODER embeddings can be used for embedding-based term normalization directly or feature for machine learning, and CODER can be fine-tuned like other PLMs.

- CODER is the first cross-lingual medical term representation supporting English, Czech, French, German, Italian, Japanese, Portuguese, Russian, Spanish, Dutch, and Chinese.

Training with relations injects medical knowledge into embeddings and aims to provide better machine-learning features. We evaluate CODER in zero-shot term normalization, semantic similarity, and relation classification standards, demonstrating that CODER outperforms various state-of-the-art biomedical word embeddings, concept embeddings, and contextual embeddings. Zero-shot term normalization is a technique for automatically normalizing medical terms to a standardized format without the need for training data in the target language. This is achieved by leveraging cross-lingual word embeddings and a bilingual dictionary.

CODER learns term representations by maximizing similarities between positive term-term pairs and term-relation-term pairs from a KG. The KG records relation triplets (h, r, t) , where the head concept h and the tail concept t are from concept dictionary D , and $r \in R$ is the relation between them. The similarity between two terms e_i and e_j in a mini-batch is defined as:

$$S_{ij} = \cos(e_i, e_j) \quad (2.1)$$

CODER also learns KG embedding inspired by semantic matching methods like DistMult and ANALOGY, which aim to approximate " $M_{r^{ht}}$ ". The motivation is to learn a better similarity function via relations. Consider two triplets (h_0, r, t_0) and (h_1, r, t_1) with the same relation type, h_0 and h_1 being semantically similar (e.g., similar diseases) may suggest that t_0 and t_1 are also semantically similar (e.g., similar drugs). We can define the term-relation-term similarity between relations as shown in the equation below:

$$S_{ij}^{rel} = \cos(M_{r_i}^T e_i, e_j) = \frac{e_i^T M_{r_i} e_j}{\|M_{r_i}^T e_i\| \|e_j\|} \quad (2.2)$$

Where $M_{rt} \in \mathbb{R}^{I \times I}$; and $\|\cdot\|$ is the L2-norm.

UMLS Meta-thesaurus

We leverage the UMLS to build the training dataset. The UMLS Metathesaurus contains medical concepts integrated from different lexicon resources. Each concept has a Concept Unique Identifier (CUI) with multiple synonymous names (terms) potentially in multiple languages. For example, concept C0002502 has the terms: Amilorid, Amiloride, producto con amilorida, etc. Each concept has been assigned one or occasionally multiple semantic types.

2.3. Challenges and Limitations in the Existing Subgraph Retrieval Phase 47

For example, the semantic types of C0002502 are Organic Chemicals and Pharmacologic substances.

The UMLS also provides related information between medical concepts in the form of triplets, such as (C5190625, C0002502, (“CHD”, “is a”)). “CHD” is the relation type. Moreover, “is a” is the detailed relationship attribute in the UMLS. Some triplets only contain the relation type or the relationship attribute. Therefore, we concatenate the names of the relation type and the attribute as relation labels of triplets. CODER is trained using the UMLS 2020AA release, which contains 4.27 M concepts, 15.48 M terms, and 87.89 M relations with 127 semantic types, 14 relation types, and 923 relationship attributes.

Training settings of CODER

We train CODERENG[11] initialized from PubMedBERT[45] for the English version with 100K training steps. We also train cross-lingual version CODERALL initialized from mBERT with 1M training steps. Terms are encoded by [CLS] representation. A batch size of $k = 128$, relation triplets, and gradient accumulation steps of 8 are used for training. We set the count of repeat triplets in each mini-batch ≤ 8 . The maximal sequence length is 32 since the terms are short. We use AdamW as the optimizer with a linear warm-up in the first 10000 steps to a peak of $2e-5$ learning rate that decayed to zero linearly. The weight is set to 1 to balance term-term loss and term-relation-term loss.

Medical term normalization tasks

They evaluate term normalization on three datasets in different languages. Term normalization tasks are evaluated in a zero-shot setting, i.e., no training datasets are provided. Cadec, PsyTar. Cadec and PsyTar are two English medical term normalization datasets. Cadec contains 6754 terms and 1029 standard terms as normalization targets. PsyTar contains 6556 terms, which are mapped to 618 standard terms. We use the data splitting from and top-k accuracy as the metric, as shown in Figure 2.16.

2.3 Challenges and Limitations in the Existing Subgraph Retrieval Phase

Structured KG is more popular than KG; Language Models do not capture the semantic meaning of the same context with billions of parameters. While

| Embedding | Cadec | | PsyTar | |
|----------------------|--------------|--------------|--------------|--------------|
| | acc@1 | acc@3 | acc@1 | acc@3 |
| GoogleNews-vectors | 39.15 | 52.41 | 29.75 | 47.52 |
| Wiki-pubmed-PMC | 35.04 | 47.14 | 23.97 | 36.02 |
| BioNLP-2016-win-2 | 35.47 | 48.30 | 25.29 | 37.70 |
| BioNLP-2016-win-30 | 38.78 | 51.64 | 28.27 | 43.26 |
| BERT-base-cased | 19.64 | 25.65 | 15.16 | 19.84 |
| BioBERT | 3.62 | 17.62 | 7.64 | 10.98 |
| ClinicalBERT(Huang) | 17.04 | 23.25 | 13.97 | 19.40 |
| ClinicalBERT | 13.77 | 18.40 | 11.12 | 15.47 |
| SciBERT | 12.50 | 17.95 | 11.04 | 15.12 |
| BlueBERT | 14.64 | 20.62 | 13.24 | 18.07 |
| PubMedBERT | 11.50 | 15.88 | 9.76 | 13.78 |
| SapBERT | 54.05 | 71.42 | 52.45 | 66.04 |
| CODER _{ENG} | 59.77 | 76.24 | 53.92 | 71.12 |
| CODER _{ALL} | 56.02 | 72.11 | 43.86 | 60.20 |

Figure 2.16: Acc@k for different embeddings in Cadec and Psy-Tar datasets. Contextual embeddings report results using the average pooling representation.

retrieving the entire Knowledge Graph is quite challenging concerning the size and memory issues. During the reasoning on the whole KG, inferring the answer to the question takes time, affecting the reasoning phase, which causes finding the incorrect solution.

The subgraph retrieval part is identical to the work of QA-GNN, GreaseLM, and Dragon. They extract a relevant subgraph G_s from the KG via an entity linking tool and get a (text, local KG) pair. We want each pair’s text and KG $G(V; E)$ to be (roughly) semantically aligned so that the text and KG can mutually inform each other and facilitate the model to learn interactive reasoning between the two modalities.

There are several challenges and limitations of existing subgraph retrieval methods:

1. First, in structured Knowledge Graphs, the size of the retrieved graph is very challenging because If the retrieved graph is too small so the chances of missing the answer because of the small size, and if it is retrieved large one, that can cause to introduce the noise that may effect on the reasoning phase [46]. Earlier methods use different strategies like retrieving top-K (Top 100 relevant nodes) entities from the triple-store (head, relation, tail). This approach can also retrieve some off-topic entities that do not belong to the question shown in figure 2.3, as mentioned in the QA-GNN [14].
2. Second, the issue about the weak internal intermediate node or entities,

which means making or establishing a connection with the intermediate node. Suppose when we need to perform a search in space, we need to explore the space, and our search starts from topic entities and explores other paths until we find the desired tail or endpoint in the space; if there is no answer in the space or the middle path P between the entities, so which path is needed to select from the space for inferring answer to the question. So, the fact that there is no correct answer for the intermediate process is called the state of Weak Internal Media Supervision, which is also faced in the existing studies [46].

This challenge can be seen in the dataset [46], only the question and its correct answer are given, and there is no answer to the intermediate nodes of which multi-hop process to find the correct answer. (Intermediate node answers missing). Therefore, the extracted subgraph G_s ignores some intermediate concept (entity) nodes and edges. In such cases, the subgraph does not contain a complete chain of reasoning.

3. Third, an Incomplete subgraph G_s is retrieved without complete information about the given entities and relations or the neighbour nodes, etc. To deal with such issues, we need to use the graph reconstruction technique that re-build the graph as discussed in the Dynamic relevance graph network (DRGN) [46] and the **case-based reasoning's CBR** [47]. The DRGN [46] models frequently cannot reason over paths when there is no direct connection between the involved concepts.
4. Subgraph retrieval can also be limited by ambiguity. It can be challenging to determine the exact subgraph to be retrieved, especially in cases where multiple subgraphs match the query pattern.

While finding the chain of reasoning in QA is challenging in general, here, this problem is more critical when the KG is the only source, and there is a missing edge; as shown in Figure 2.17, the KG sub-graph misses the direct connection between **Guitar** and playing an instrument (green arrow) in Figure 2.17. As pointed out, previous models are not sensitive to the negation words and consequently predict opposite answers for the issue of considering question semantics [8].

This image shows the weak internal connection or missing edges between the nodes and is taken from a paper **DRGN** [46].

5. It becomes a significant challenge to reduce the computation cost of nodes and the memory issue if we can retrieve the exact portion of the sub-graph from an extensive knowledge graph.

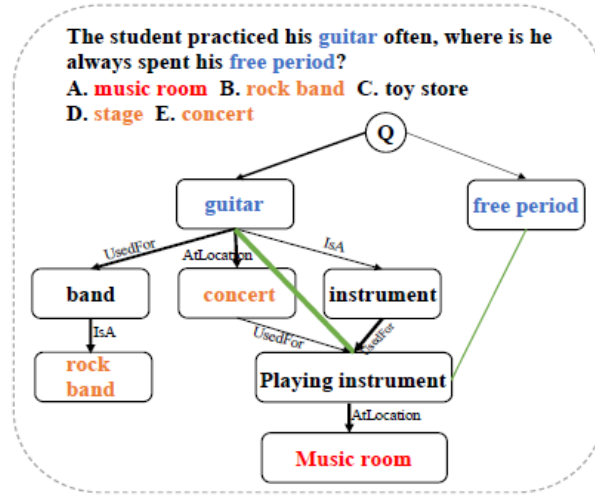


Figure 2.17: The weak or missing edges between the nodes image taken from [9].

The solutions proposed by QA-GNN [14] somehow deal with the issues, but the question here is how to retrieve the relevant portion of the subgraph G_s from the significant knowledge graph. Through the Relevance scoring, they compute the relevance of KG nodes conditioned on the given QA context, but how to measure and explore the space for retrieving the relevant nodes from the large KGs triples. The recent research QA-GNN, GREASELM, and the Dragon model is the first work to deal with the negative questions. QA-GNN improves the reasoning under negation by adding the QA global node to the graph. However, the challenge still exists.

These are significant challenges mentioned above. However, these challenges are the issues that the various methodologies currently being researched are trying to solve. Our work presents a novel plug-and-play subgraph retrieval framework model for these challenges. This efficient method combines the Language Model and Structured Knowledge Graph to retrieve the controlled size and relevance subgraph, preserving the semantics and inferring the correct answer to the given question. A detailed explanation of decoupled subgraph retrieval is in chapter 03, with a graphical understanding of the methodology.

2.4 Dense Retrieval

Text-based information retrieval refers to searching for and retrieving information from unstructured or semi-structured text sources, such as documents,

articles, or databases. This is typically done using a search engine or database system that indexes the text and allows users to search for keywords or phrases. A text-based information retrieval search results are ranked based on relevance scores, with the most relevant results appearing at the top of the list. This approach to information retrieval is widely used in many industries, including academic research, e-commerce, and legal research.

Generally, in the QA system, the first step is the retrievals phase, in which the system identifies the relevant document from the set of large documents or corpora. The second is the reasoner phase, in which the reasoner performs the reasoning and identifies the exact answer from the set of retrieved or identified documents, considered the exact answer to the question.

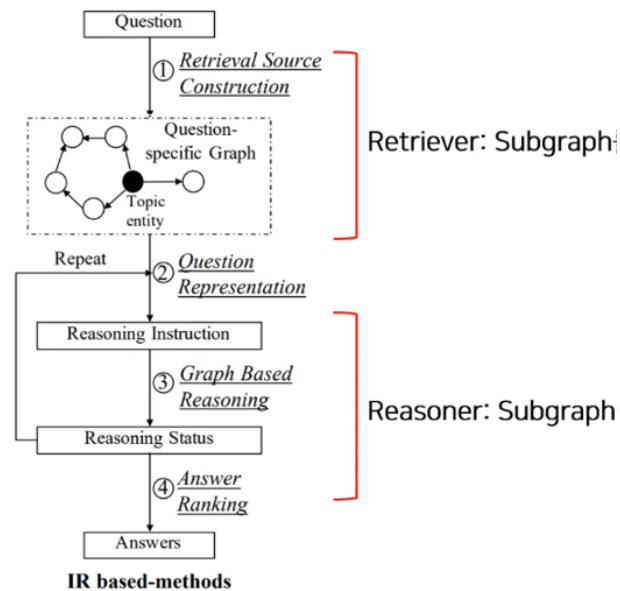


Figure 2.18: Information Retrieval and Reasoner phase

Information Retrieval (IR)

Subgraph retrieval is the very complex phase to retrieve the relevant portion of the subgraph G_s from the Structured Knowledge Graph KG. Still, it is also crucial to find the exact entities and relations to construct the subgraph G_s . For traversing the graph and performing reasoning, we have some essential preliminary steps, like identifying entities and relationships from the knowledge base structure or the triple store constructed in the offline pre-processing phase of the dataset USMLE.

The entities extracted from the question are called the topic entity. Our starting point in the dense space setting is these topic entities, which we use to find answers. The right way to find the correct answer is by calling the part related to the topic entity in the question and searching for the relation and tail entities of the topic entity. In addition, Logical Base Question Answering is a task to calculate or find a correct answer entity through a KB Base (triples) for a factoid question in natural language with a definite answer.

2.4.1 Language Model for Dense Passage Retrieval

Recent research shows most researchers try to make the retrieval phase more efficient than the existing one. They are trying to find new ways of information retrieval or improve the existing methods in accuracy and efficiency. In the Dense Passage Retrieval (DPR) [26], the retrieval phase of the system selects the passage or paragraph closer to the answer and then extracts the passage using NLP and retrieves the exact answer to the given question. The studies in DPR [26] show improved retrieval components in open-domain QA. The aim of DPR is to retrieve and index the passage from the set of passages M , and transform them into a low-dimensional and in continuous space.

Searching and indexing the relevant document from a set of documents can be done in two ways. **(1) TF-IDF and (2) BM25 model.** TF-IDF is a mathematical framework and very traditional and frequency-based method that shows the importance of each word concerning documents. TF-IDF has some restrictions related to the size of the given documents and the length of the terms matrix; also, it deals with individual documents at a time.

BM25 model for documents retriever: BM25 is a probabilistic retrieval framework for the Open-domain QA (OpenQA) [13]. The system aims to answer based on many documents like Wikipedia. In the Open-domain QA, the retrieval phase is considered the system's backbone because the retrieval function retrieves the set of documents based on the given query, retrieves those documents containing the query terms, and ranks them according to

their similarity scores. BM25 is a popular method that retrieves documents or passages concerning the given query.

The model can efficiently retrieve the top K passage relevant to the input question. DPR [26] uses dual dense encoder E_p for passage and E_q for the question. E_p transforms the given text into the dimensions of a real-valued vector and builds an index for all passage \mathbf{M} for the retrieval phase. At run time, the different encoder for question E_q , E_q maps the input question to a d-dimensional vector. It retrieves the K passage closest to the question vector from the index passages M .

$$\text{sim}(q, p) = E_Q(q)^T E_P(p) \quad (2.3)$$

Equation 2.3 Similarity between the question E_Q and passage E_P using the dot product of their vector representations.

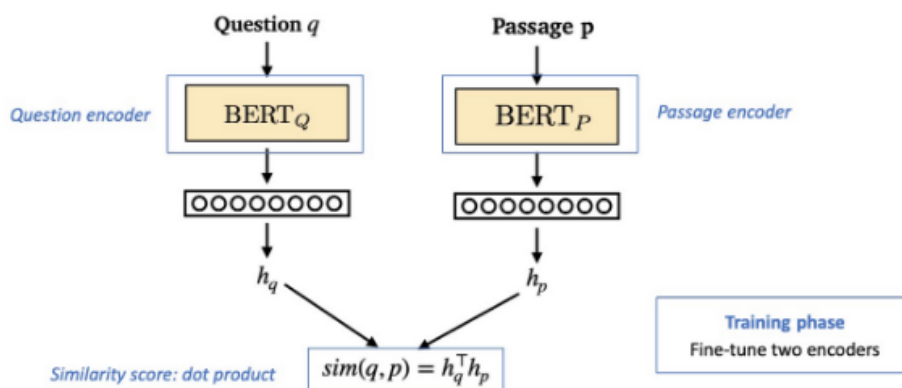


Figure 2.19: Graphical representation of the equation 2.3, How to measure the similarities between the embedding of question E_Q and the passage E_P .

2.4.2 Differentiate between Single-hop and Multi-hop reasoning

In the reasoner phase, we need to use different attention mechanisms, **Graph Attention Network GAT** or Graph neural network GNN, paying attention to inferring the answer to the question. For searching the answers, we need to traverse the graph by going from entity to entity and relation to relation, sometimes, our answer is directly linked to the question, and we need to use one hop or one-step traversal of the graph to find the exact answer to the questions.

A one-hop question means a question that needs to be breathed, traversed, or reasoned only once.

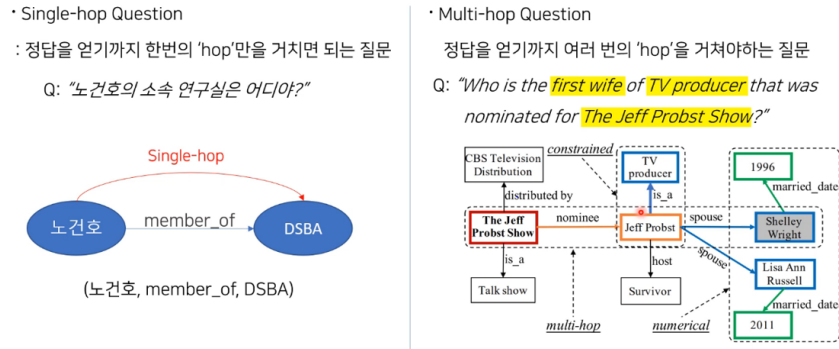


Figure 2.20: Difference between single and multi-hop reasoning for inferring answer to the given question.

For example, the laboratory **DSBA** that I belong to which is directly connected to me at once through the relation (member of), as shown in the following figure 2.20. Hence, the answer can be found in these simple forms beyond easy. But in the multi-hop question, the answer is not directly connected with the topic entities, and we need to go far in the search for the answer, as shown in the Figure. In multi-hop, a distinct critical advantage is finding answers to complex questions by crossing several objects through relational connections between entities. This is called multi-hop question reasoning.

A **multi-hop question** on the right of the Figure, we have the following question: Who is the first wife of a TV Producer nominated for the **Jeff Probst show**? For the multi-hop, there are many parts that the graph needs to pay attention to or fetched from the knowledge base. The correct answer must be searched through multiple relations and entities to reach the answers.

Chapter 3

Our Solution

3.1 Stanford UMLS Knowledge Graph

Bio-Medical Knowledge Graph (UMLS)

The Stanford UMLS Knowledge Graph (KG) is a large-scale knowledge base created by Stanford University that combines information from the Unified Medical Language System (UMLS) with other sources to represent biomedical knowledge comprehensively. The UMLS KG triplet store is a database system designed to store and manage the triplets that make up the UMLS KG.

The Unified Medical Language System (UMLS) Knowledge Graph (KG) contains information about a wide range of biomedical fields, including but not limited to:

- **Anatomy:** Information about different body parts, organs, and systems.
- **Pathology:** Information about diseases, disorders, and conditions.
- **Pharmacology:** Information about drugs, their uses, interactions, and side effects.
- **Medical procedures:** Information about surgical and non-surgical procedures.
- **Physiology:** Information about the body's functions and its systems.
- **Genetics:** Information about genes, genetic disorders, and inheritance.

The knowledge graph provides a comprehensive representation of biomedical knowledge and is a valuable resource for the biomedical community.

3.2 Knowledge Graph Embeddings

Graphs are collections of facts in the form of ordered triples (h, r, t) , where entity h is related to entity t by relation r . Because knowledge graphs are often incomplete, the ability to infer unknown facts is a fundamental task (link prediction). A series of recent KGE models approach link prediction by learning embeddings of entities and relations based on a scoring function that predicts a probability that a given triple is a fact. Knowledge Graph Embeddings represent entities and relationships in a knowledge graph as dense vectors, also known as embeddings, in a high-dimensional vector space. This method aims to capture the complex relationships between entities and their attributes in a compact and continuous representation that can be used for various downstream tasks, such as link prediction, knowledge graph completion, and information retrieval.

Benefits of KG Embeddings

The main idea behind dense knowledge graph embeddings is to train a neural network on a large corpus of text and use the learned representations of entities and relationships in the knowledge graph as embeddings. This can be done using various methods, including TransE, DistMult, and ComplEx. The resulting embeddings capture the complex relationships between entities and their attributes in a continuous and high-dimensional vector space, which can be used to perform various machine-learning tasks, such as classification, clustering, and recommendation.

Knowledge Graph Embeddings offer several benefits for representing and analyzing graph data:

- **Compression:** By mapping the nodes and edges of a graph to dense vectors, dense graph embeddings provide a compact and continuous representation of the graph structure and relationships, making it easier to store, manipulate, and analyze large graphs.
- **Interpretability:** Dense graph embeddings provide a high-dimensional continuous representation of graph data, which can be visualized and interpreted more easily than sparse graph representations. This can be useful for exploring and understanding the structure of a graph.
- **Transfer learning:** Dense graph embeddings can transfer knowledge from one graph or task to another. For example, pre-trained embeddings can be fine-tuned on a new task or used to initialize the weights of a model for a new graph.

- **Scalability:** Dense graph embeddings can be computed and stored efficiently, making it possible to handle large-scale graphs.
- **Improved performance:** Dense graph embeddings have been shown to improve performance on graph-related tasks, such as node classification, link prediction, and graph clustering, compared to traditional graph representation methods.

3.3 Reasoning over Knowledge Graphs

Subgraph G_S construction is a similar phase in the QA-GNN, GREASELM, AND DRAGON. While extracting the entities from the question or the topic entities in the Dragon, it uses an additional phase. It used an entity-linking extractor to extract the entities from the question. Then, it tried to use the QA-GNN methodology to retrieve from the KG and compute the likelihood between the QA context and the retrieved entities. As shown in Figure 2.3, the more relevant entities concerning their relevance score can be linked near the QA context and similarly to other entities. In the QA-GNN, they have a typical working graph and then perform joint reasoning on them to predict the answer to the given input question.

Figure 2.3 shows the entities retrieved from the existing approach; some are off-topic, and some are semantically irrelevant to the given QA context. Like **banks**, those entities are considered outliers or noise that impact the reasoning process to find the exact answer. For example, nodes **“holiday”** and **“riverbank”** are off-topic; **“human”** and **“place”** are generic entities. These irrelevant or off-the-topic nodes may result in over-fitting or introduce noise that can affect the reasoning phase while retrieving the exact answer to the question.

Discrete Subgraph retrivals

The current retrieval phase retrieves the discrete subgraph from a knowledge graph and involves finding a subgraph disconnected from the rest. It can be done using graph traversal algorithms such as depth-first search or breadth-first search to identify all the nodes and edges that belong to the subgraph. Alternatively, graph database query languages like SPARQL can retrieve the desired subgraph. The specific approach may depend on the knowledge graph’s type and structure.

The major drawback of discrete subgraph retrieval from a knowledge graph (KG) is that it may not capture the full context or relationships between entities in the KG. Discrete subgraph retrieval involves extracting a subset of nodes

and edges from the KG that match a particular query, which may not provide a complete picture of the surrounding entities and their relationships. This could limit the accuracy and effectiveness of downstream applications that rely on KG data.

Dense Subgraph retrieval

Finding a dense subgraph in a knowledge graph is an active area of research in graph theory. It has many real-world applications, including community detection, social network analysis, and recommendation systems. One common approach to finding dense subgraphs is to use algorithms that measure the density of the subgraph, such as the modularity metric. First, choose a starting node in the graph space and initialize a set to store the nodes and edges of the subgraph. Next, use a density metric, such as modularity or clustering coefficient, to evaluate the density of the subgraph.

Finally, add neighboring nodes to the subgraph and re-evaluate the density metric until the density no longer increases or a stopping condition is met. Finally, return the final subgraph as the solution to the problem. This involves formulating a query specifying the subgraph density to be retrieved and executing the query against the graph database. Using graph database (Like GraphDB), query languages like SPARQL can retrieve a dense subgraph. Again, this involves formulating a query specifying the subgraph density to be retrieved and executing the query against the graph database.

Retrieving dense subgraphs from a knowledge graph (KG) can have several benefits, including:

- **Capturing rich context:** Dense subgraphs contain many interconnected entities, providing a complete picture of the surrounding entities and their relationships. It can help downstream applications better understand the context and make more informed decisions.
- **Identifying important entities:** Dense subgraphs often contain important entities that play a vital role in the KG. Identifying these entities can provide valuable insights into the structure and function of the KG.
- **Efficient processing:** Retrieving dense subgraphs can be more efficient than retrieving individual entities or discrete subgraphs. It is because many algorithms for subgraph retrieval are optimized for dense subgraphs, allowing for faster processing times and lower computational costs.

Overall, retrieving dense subgraphs can unlock the full potential of KGs and improve the accuracy and effectiveness of downstream applications.

3.3.1 Subgraph Retrieval Enhanced Model for Multi-hop Knowledge Base Question Answering

This paper [12] presents the idea of a **Subgraph retriever (SR)** that is very useful and easy to replace in the existing work of **QA-GNN ([14])**, **GreaseLM [8]**, and the **DRAGON [9]** subgraph retrieval phase. The subgraph retrieval SR is a plug-and-play framework independent of the following reasoning phases. It has a significantly better subgraph retrieval phase than the existing methods mentioned in the QA-GNN [14], and the DRGN [46]. This **SR** extracts the full multi-hop topic-centric subgraph from the large KG and also produces the control graph size with the help of **personalized PageRank (PPR)** scores of entities, like by setting the threshold which is set for the selection of the entities and paths.

In the existing work on each step, the **LSTM-based retriever** selects new relations, a path relevant to the question or the topic entities, after the GNN-based reasoner determines which tail entities of the new relations should be expanded into the subgraph. The result of this methodology is to perform reasoning on the partial graph that is missing some parts or nodes relevant to the given question.

Approach

Subgraph Retrieval SR is developed as an **efficient dual encoder model**, which can expand paths or relations to induce the subgraph and stop the path's expansion automatically. The goal of the SR is to find the sequence of links or connections with the highest probabilities and then induce the tree. For deducing the answer from the merged graph, we use **GraphNet or Neural state Machine NSM** to infer the responses from the retrieved subgraph. This decoupled or separable retrieval and reasoner can only reason on the final retrieved subgraph instead of partial or the large one, which can be considered costly concerning time and computation of the relation and node awareness.

Moreover, this **DSR** will also construct the controlled subgraph concerning the size that is not very small or too large and the missing intermediate node connection because it explores those sequences of paths that can fulfil the threshold criteria.

Problem Definition

The information in the KG is organized in the form of triples which contains the head, tail, and relations. $G = (e, r, e0)$ where the E represents the set of entities and set relations, respectively. As per the given question q , KG-based retrieval aims to figure out the answer A_q to the question from the set of entities

E of Knowledge Graph KG. The entities E_q mentioned in the given question \mathbf{q} are called topic entities, which are already given. In our dataset USMLE, we have complex questions where the answers are not directly available, so we need multi-hop reasoning to obtain the answers of the topic entities.

In the probabilistic formulation, we aim to maximize the probability of answers a to the question by measuring the probability of answer a given by graph G and question q , respectively.

$$p(a/G, q) = \sum_G P\phi(a/q, G_s)P\theta(G_s/q) \quad (3.1)$$

Instead of performing direct reasoning on the entire G , we need to retrieve the subgraph G_s from the Knowledge graph G and then infer the answer a . According to the given equation 3.1, we have two modalities, the first is to retrieve the portion of subgraph G_s as per given the question q , and the second one is to infer the answer given by the retrieved subgraph G_s and the question.

The goal is to find the optimal parameters θ (subgraph retriever G_s conditioned by question q) and ϕ (inferring the answer condition by question q and the G_s) that maximize the log-likelihood of training data as shown in equation 3.2.

$$L(\theta, \phi) = \max_{\theta, \phi} \sum_{(q, a, G) \in D} \log \sum_G P\phi(a/q, G_s)P\theta(G_s/q) \quad (3.2)$$

D contains the whole training dataset, and here we present the concept of decoupled phase for the retrieval $p\theta$ and the reasoner $p\phi$. Both parts (retriever $p\theta$ and reasoner $p\phi$) can be optimized respectively, as shown in equation 3.3.

$$L(\theta, \phi) = \max_{\theta, \phi} \sum_{(q, a, G) \in D} \log P\phi(a/q, G_s)P\theta(G_s/q) \quad (3.3)$$

Overview of Subgraph Retrieval

Figure 3.2 demonstrates how subgraph G_s is constructed using the topic entities \mathbf{E}_q from the question q using the relations r or path. Topic entities \mathbf{E}_q are our starting points, and we expand routes from the topic entities and induce a corresponding tree. When we have generated trees from each topic entity, we merge those induced trees into the standard unified graph using the union techniques. After all the work, as a result, we have a standard graph. It is used for the next reasoner phase to perform reasoning to infer the answer to the given question \mathbf{q} instead in the initial stage on the KG without filtering anything.



Figure 3.2: Illustration of the subgraph retrieving process. We expand a path from each topic entity, induce a corresponding tree, and then merge the trees from different topic entities to form a unified subgraph [12].

3.3.2 How to expand paths from topic entities?

We need to input the topic entity and its connected path for an inferring path to the subgraph. The topic entity is a node representing the starting node in a knowledge graph, and the path represents the edges following from the starting node to form a subgraph. Through this function, we got a set of nodes and a set of triples from the subgraph. To find the candidate relation from the given paths, we use the topic entity and path and deduce the relations that can be formed between the leaf nodes of the subgraph specified by the path and the topic entity. The function then filters out relations with a namespace of KG or common. Finally, the function returns the set of candidate relations as a list of strings.

Path Queries

For example, the path expedition starts from the starting points in the search space, like the topic entities mentioned in question (**Turing Award**, **Canada**). In the case of our dataset **USMLE**, we already have all set of entities and relationships that are in the form of triples (**head**, **relation**, **tail**). Our goal is to identify the sequence of paths or relations $(r_1, r_2 \dots, r_n)$, the question q can indicate the intermediate relations excluding the tail entities in the search space, here we have a concept of partial path $p(t) = (r_1, r_2 \dots, r_t)$ that are retrieved at time t to the questing entities **Eq**. For each partial path at each time step t , a tree can be induced from the retrieved path $p(t)$ and fill the intermediate or missing entities along those paths. For example: $T(t) = (E_q, r_1, r_2 \dots, r_t), E_t)$.

With the help of a partial path, we can derive the entities E_t , each entity E_t considered as the head entity, and a relation with the head entities can derive multiple tails entities. This task is similar to the tail entities prediction with the help of a predefined head and relation. But the question is how to select the following relation from the all-neighbor relations (union of relations, like in a **WordNet**, we have 29 relations) of head entities E_t . The solutions are

to measure the relevance of each relation r to the question by taking the dot product between their embeddings, as shown in equations 3.4.

$$s(q, p) = f(q)^T h(r) \quad (3.4)$$

In equation 3.4, both f and h are instantiated by **RoBERTa** [22], which represents the same concept as used in the **Dense passage Retriever** DPR [26] a dual encoder-based model. Instead of the BERT [3] encoder, we use a **dual Roberta base model**, one for relation and one for the question. The mechanism for passing input to the model is similar to DPR [26], we input the question and relation to the Roberta model, and it returns the **[CLS]** token as the output embeddings and the measure of the similarities between the representation of question **R_q** and relations R_r .

Conjunctive Queries

We aim to find relevant paths in a knowledge base (KB) or KG to answer a question. We start with an initial set of candidate paths, a list of paths with their corresponding scores. Then, at each iteration, we calculate the next set of candidate relations for each path based on the current topic entity and the path history, as shown in equation 3.5.

With partial path $\mathbf{p}(\mathbf{t})$, which is expanded for predicting the tail entities, for updating the embedding of question $f(q)^T$, we use the embedding of each partial relation $\mathbf{p}(\mathbf{t})$ at time \mathbf{t} and concatenating them to the original question q as shown in equation 05.

$$f((q)^T) = \mathbf{RoBERTa}([q, r_1, r_2, r_t]) \quad (3.5)$$

After the updating embedding of the question $f((q)^T)$, we can modify equation 3.4 and replace the embedding of the previous embedding of question \mathbf{q} with the updated one $f((q)^T)$ and then measure the similarities with the embedding of relations as shown in the **equations 3.6**.

$$s(q^t, r) = f((q)^t)^T h(r) \quad (3.6)$$

The question \mathbf{q} here is when to stop the expansion of the relations \mathbf{r} . The probability $P(r/q^t)$ of the relations given by the questions at time \mathbf{t} can be formalized in equation 3.7. Then, we score each path-relation pair based on how well the pair matches the question and retains a certain number of the highest-scoring path-relation pairs as the new set of candidate paths. The process continues until either no more candidate paths are left or Until we Reach the virtual End keyword, as shown in equation 3.7, the maximum number of hops is reached, or the required number of result paths is found.

$$P(r/q^t) = \frac{1}{1 + \exp(s(q^t, \mathbf{END}) - s(q^t, r))} \quad (3.7)$$

In equation **3.7**, we use the **END** token as the virtual relation that is considered as the threshold. The similarity score $s(q^t, \mathbf{END})$ shows the similarity measure between the question and the with **END**. The probability of each relation $p(r|q^t)$ given by question at time t is larger than the **set threshold 0.5** then we consider as the relevant relation; otherwise, we discard them and select another relation until we reach $p(t) = (r_1, r_2 \dots, r_n)$ and repeat the process for all relations. As a result, we select the top one relation $p(r|q^t) > \mathbf{0.5}$ whose score is higher from the ranked relations list and predict the tails entities.

If the probability of relation is $p(r|q^t) > \mathbf{0.5}$ is not larger than **0.5**, the expansion automatically stops. Finally, the probability of a path p given the question q can be computed as the joint distribution of all the relations $(r_1, r_2 \dots, r_t)$ in the path p as shown in equation **07**.

$$P\theta(p/q) = \prod_{t=1}^p P(r_t/q^t) \quad (3.8)$$

As in equation **3.8**, $|p|$ shows the number of relations r in p , the time $t = 1$ shows the selection at the topic entities, and $t = |p|$, denotes the last none-stop relations selections. The selection of the $top - 1$ relevant path cannot be guaranteed the right one from all paths; we use another technique called **Beam search**. We perform a Beam search with a beam size of **2** and get $top - k$ paths p concerning time t . For each topic entity Eq from question q , through Beam search, we obtain K paths which result in nK paths in total by n topic entities and every nK path link with nK instantiated tree as shown in Figure 3.3.

In the given figure, the goal is to find the sequences of the relations $(r_1, r_2 \dots, r_n)$ with the highest probability instead of an exact-match search because it is an expensive approach, so we use another approximated searching technique called beam search, Beam search expands the nK path and those nK path inducing the trees.

3.3.3 Subgraph Construction Through Expanded Paths

We induced trees from the expanded nK path; for each topic entity, we have an nK path, and every path has a tree representing the head and tail entities linked through the relation from the expanded nK path. Simple, we use a mathematical operation called **UNION**. We combine top-K expanded trees from one topic entity into a single subgraph. Ultimately, we merge the

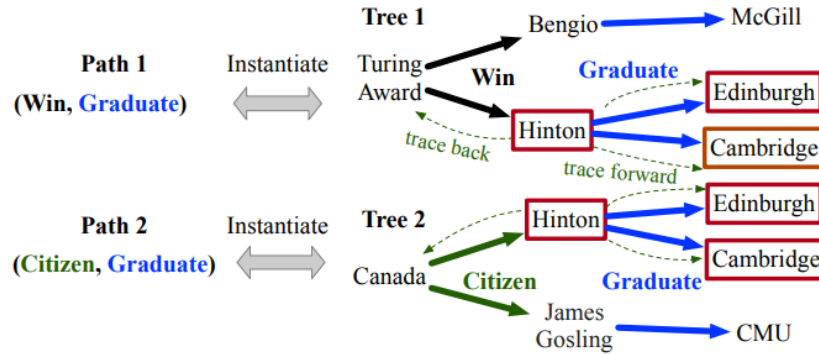


Figure 3.3: Trees inducing through the selected paths as shown in the image, taken from the work [13].

same entities from different N subgraphs to induce a common subgraph called the final subgraph. For the second reasoning phase in the model, we use that final subgraph to infer the answer to the given question q , as shown in the third part of the images below.

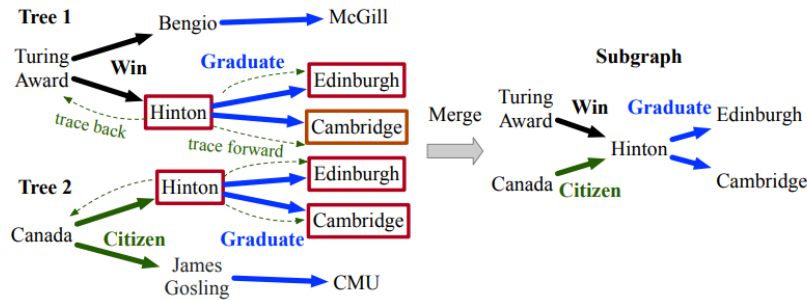


Figure 3.4: Trees merging into a unified common graph taken from

Forward and Backward Search

Through standard graphs or merged entities, we can trace back from the tail node or leaves node to the root or topic entities and vice versa for the trace forward. For example, in Figure 3.2, we have a given input question “**Where did Canadian citizens with Turing Award graduate?**” with two topic

entities “**Turing Award**” and “**Canada**”, there are two paths expanded from topic entities (**Win, Graduate**) and (**Citizen, Graduate**) and then merged the trees that are induced by them into common unified subgraph as shown the figure 3.4.

3.3.4 Subgraph Retrieval and Training

Subgraph retrieval and training are two critical components of subgraph retrieval-enhanced models for multi-hop reasoning in knowledge graphs (KGs). Subgraph retrieval involves identifying relevant subgraphs from the KG that can be used to support multi-hop reasoning. It involves selecting a subset of nodes and edges from the KG connected to a starting entity and forming a subgraph relevant to the query being asked. The retrieved subgraph is then used to perform multi-hop reasoning to infer new knowledge or answer the query.

While the training in subgraph retrieval-enhanced models involves using Language Models and Machine Learning algorithms to learn how to retrieve relevant subgraphs from the KG, this typically involves training a neural network or machine learning model to predict which subgraphs are most relevant to a given query. Then, the model is trained on a dataset of queries and their corresponding subgraphs using supervised or reinforcement learning techniques.

By combining subgraph retrieval and training in a single model, subgraph retrieval-enhanced models can learn to effectively retrieve relevant subgraphs from the KG and use them to support multi-hop reasoning. This can improve the accuracy and effectiveness of reasoning in KGs and enable more sophisticated downstream applications.

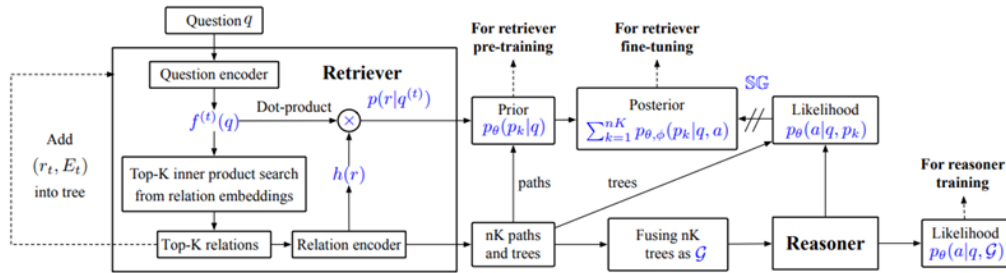


Figure 3.5: Subgraph Retrieval SR and training phases.

Given the input question q “**Where did Canadian citizens with Turing Award graduate?**” with two topic entities such as “**Turing Award**” and “**Canada**”². The goal of subgraph retrievals is to find the sequence of relations.

SR iteratively expands the relations of the mentioned topic entities and then generates the **nK** paths. We pre-train the retriever based on the prior of each track and train the reasoner based on the likelihood of the subgraph fused from the nK trees. For end-to-end training, the retriever is fine-tuned on each path’s posterior, consisting of its prior and likelihood.

The ground truth subgraphs construction is not easy to retrieve, so we alternative to the weak supervision signals constructed from the given question and answering (q, a) pairs. From each topic entity Eq of a question q , we obtained all the shortest paths to each answer and considered them as the supervision signals, as the initial paths are more accessible to retrieve than graphs. Moreover, since maximizing the log-likelihood of a path equals $P|p|t = 1\log p(r_t/q^t)$ according to 3.8, we can maximize the probabilities of all the intermediate relations in a path.

To achieve the goal, we decompose a path $p = (r_1, , r|p|)$ into $|p| + 1$ **(question, relation)** instances, including $([q], r_1)$, $([q; r_1], r_2), \dots, ([q; r_1; r_2; ; r|p|1], r|p|)$, and an additional *END* instance $([q; r_1; r_2 ; ; r|p|], END)$, and optimize the probability of each instance. We replace the observed relation at each time step with other sampled relations as the negative instances to maximize the likelihood of the observed ones.

Chapter 4

Experimental setup

4.1 Dataset

MedQA-USMLE (MedQA) is a 4-way multiple-choice task containing the United States Medical License Exam questions. The dataset has 12,723 questions. Each sample contains a question, correct answer(s), and other options which require a deeper language understanding as it tests the 10+ reasoning abilities of a model across a wide range of medical subjects and topics. A detailed explanation of the solution, along with the above information, is provided in this study. MedMCQA provides an open-source dataset for the Natural Language Processing community. The MedQA USMLE dataset is not used for the experiment due to its large size and complexity in making a fair comparison. Additionally, the dataset contains additional meta information on the questions and answers.

System Configuration

We use Google Colab notebook for trim levels of testing and prototyping because it provides a free cloud-based Jupyter notebook environment that is convenient for prototyping and testing small-scale machine learning models. However, to train more extensive and complex models or handle larger datasets, we need more computing resources than are available in a typical Colab environment. In particular, training models on large datasets can require significant computational resources, including GPUs or TPUs, high-performance CPUs, and large amounts of memory. While Colab provides access to GPUs and TPUs, the available resources are limited and may need to be improved for large-scale training tasks.

In contrast, server settings can provide much larger computing resources, including high-performance CPUs, multiple GPUs or TPUs, and large amounts

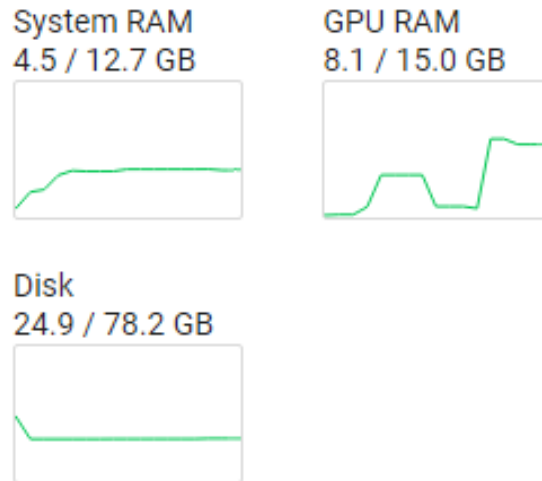


Figure 4.1: Google Colab notebook resources utilizations.

of memory and storage. These resources are typically optimized for high-performance computing tasks and can support more significant and complex machine-learning models. Therefore, for large-scale training tasks, it is necessary to use server settings that provide more computing resources than are available in a typical Colab environment.

Hardware Setup: We ran each experiment on a workstation having one Nvidia GeForce RTX3090 GPU with 24GB of dedicated memory, 64GB of RAM, and an Intel® Core™ i9-10900X1080 CPU @ 3.70GHz.

Experiment tracking: We track all our trainings with Weights and Biases10 and monitor CO2 emissions with CodeCarbon11. Moreover, we profile the neighbors’ retrieval speed with custom code.

SLURM (Simple Linux Utility for Resource Management): SLURM is a workload manager used in high-performance computing (HPC) environments to schedule and manage to compute jobs across a cluster of nodes. It provides a framework for submitting, managing, and monitoring jobs and allocating and tracking resources such as CPUs, memory, and GPUs. SLURM is open-source and widely used in academic and research computing environments. Through sbatch we submit the jobs for the executions as mentioned below.

sbatch is a command in the SLURM workload manager used to submit batch jobs for execution on a compute cluster. It specifies job requirements such

as the number of nodes, CPU and memory requirements, job time limit, input and output files, and other parameters. Once submitted, the job is added to the queue and is scheduled to run on the cluster as resources become available. The sbatch command is typically used with a script specifying the job’s details.

Pre-processing on UMLS KG

There are two ways of training the model; the first one is Online processing, which refers to the use of a model in a real-time setting, where input data is processed as it is received, and the model produces output immediately. The second one is offline processing, which refers to using a model on a dataset that is not actively used or updated.

We used the Stanford UMLS Knowledge Graph (KG) [14, 9], we downloaded the KG from the Dragon and QA-GNN, UMLS is a large-scale, structured representation of biomedical knowledge created by Stanford University. It combines information from the Unified Medical Language System (UMLS) with other sources to provide a comprehensive representation of biomedical knowledge in the form of a knowledge graph.

UMLS KG Triples

A triple in the Stanford UMLS Knowledge Graph (KG) consists of a subject, a predicate, and an object. Each triple represents a relationship between two entities in the KG. Here are some examples of triples in the UMLS KG:

| # | Subject | Predicate | Object |
|---|-------------------|----------------------|-----------------------|
| 1 | Diabetes Mellitus | has-diagnosis | Diabetes |
| 2 | Insulin | "is-a-drug-used-for" | Treatment of Diabetes |
| 3 | Fever | is-a-symptom-of | Influenza |
| 4 | Heart Attack | is-treated-with | Aspirin |
| 5 | Breast Cancer | has-treatment | Surgery, |
| 6 | Breast Cancer | has-treatment | Chemotherapy |

Table 4.1: Example of Stanford UMLS KG Triples

Here in the given triples, the subject and object are used as the nodes or entities in the graph, while the relations are considered as the edges in between the subject and object. These are just a few examples of the relationships represented in the UMLS KG. The knowledge graph provides a rich representation of biomedical knowledge and is a valuable resource for the biomedical community.

Triplet Store Construction

In the offline preprocessing phase, the first step is to store the extracted triplets in the triplet store or organize them in a helpful representation format. This involved selecting a suitable database management system, designing the schema, such as Tab-separated values or CSV for the store, and optimizing for efficient querying and retrieval of information. After the construction of the triplet store, it was populated with the extracted triplets, and various algorithms were used to enhance its functionality, such as entity disambiguation, relationship extraction, link prediction etc.

During preprocessing and construction of triplet files, we observe from the related work published by the same author that and having similar subgraph retrievals phases like QAGNN, GreaseLM, and Dragon. For example, while analyzing the preprocessed data from Dragon and QAGNN, In QAGNN, we have less number of relations; while exploring the other processed data from GreaseLM and Dragon, we observe that the processed KG in the QAGNN is very small, containing 15 relations, while on the other hand, the size of knowledge graph KG in the Dragon is substantially large as compared to QA GNN. For example, in the QA GNN, we only have 15 types, while in the Dragon, we have 98 relations, as shown in Table 4.2.

| Relations | Head Nodes | Tail Nodes |
|-----------|------------|------------|
| 99 | 220737 | 294385 |

Table 4.2: Unique UMLS Knowledge Graph features (Head, Tail, and Relations).

Knowledge Graph Embedding Models - PyKEEN

TransE

We split the UMLS KG dataset into training, testing, and validation subsets in Open-domain question answering. We need to define our split based on the specific requirement task in practical use. Like UMLS KG dataset has been split into three subsets for training, testing, and validation. The training subset contains 80% of the data, while the testing and validation subsets each contain 10% of the data. While the pyKEEN splits split the data into training, testing, and validation subsets using the pykeen.slicing module, which provides functions for creating various types of data splits.

For graph embedding, we use the most popular methods for the medical dataset. The python library PyKEEN performs knowledge graph (KG) embedding on a set of triplets. The triplets are split into train, evaluation, and test

sets. The KG embedding algorithm used is TransE, a popular algorithm for KG embedding.

- **training:** the train set of triplets.
- **validation:** the evaluation set of triplets.
- **testing:** the test set of triplets.
- **model:** the name of the KG embedding algorithm to use (in this case, TransE).
- **stopper:** the stopping criterion for the training process (in this case, early stopping).
- **epochs:** the maximum number of epochs to run the training process for (in this case, 100).
- **dimensions:** the number of dimensions of the embedding vectors (in this case, 512).
- **random_seed:** the random seed used to initialize the embeddings (in this case, 420).

The pipeline function takes several arguments:

The results of the KG embedding can be saved to a directory for later use.

DistMult

We use the PyKEEN library to perform knowledge graph (KG) embedding on a set of triplets constructed in the initial offline preprocessing section. First, we split the triplets into the train (80 %), evaluation (10 %), and test (10 %) sets using the TriplesFactory method from PyKEEN. The resulting train and evaluation sets will be used to fit the KG embedding model, while the test set will be used to evaluate the model's performance. To perform the KG embedding, The KG embedding algorithm is DistMult, a widely used algorithm for KG embedding.

The pipeline function takes several arguments:

- **training:** the train set of triplets.
- **validation:** the evaluation set of triplets.
- **testing:** the test set of triplets.

- **model:** the name of the KG embedding algorithm to use (in this case, DistMult).
- **stopper:** the stopping criterion for the training process (in this case, early stopping).
- **epochs:** the maximum number of epochs to run the training process.

Finally, the KG embedding model results are saved to a directory named "stanford_kg_transe" for future use instead of training repeatedly.

Hyper-parameters Tuning - PyKEEN

PyKeen provides several options for tuning the hyperparameters of a knowledge graph embedding model. One way is to use the built-in hyperparameter tuning functionality, which utilizes the Optuna library. It allows you to specify a range of possible values for each hyperparameter and the number of trials to run. It will then search over the parameter space to find the best hyperparameters based on a specified evaluation metric.

Another way to tune the hyperparameters is by specifying a configuration file that contains the hyperparameter settings. This allows you to set the hyperparameters and fine-tune the model manually. Additionally, PyKeen allows using other libraries, such as Hyperopt and Optuna, to perform hyperparameter tuning. It will enable the user to specify the optimizer and the search space in the configuration file. It is essential that the selection of hyperparameter values can significantly affect the model's performance, so it is generally recommended to perform a thorough hyperparameter tuning procedure before evaluating the model on a test set.

Embeddings with CODER

CodeR [48] is a cross-lingual medical term embedding system designed for term normalization. The knowledge-infused system leverages external knowledge sources and text data to generate embeddings for medical terms. The purpose of codeR is to map multiple variants of medical terms (such as synonyms, abbreviations, or different spellings) to a single normalized form. This can be useful in various medical NLP applications, such as information retrieval, text classification, or named entity recognition. The knowledge-infused approach used by codeR allows it to generate more accurate and robust embeddings than standard term embedding methods that rely only on text data. In addition, by incorporating external knowledge sources, codeR can better capture the semantic meaning of medical terms and improve normalization performance.

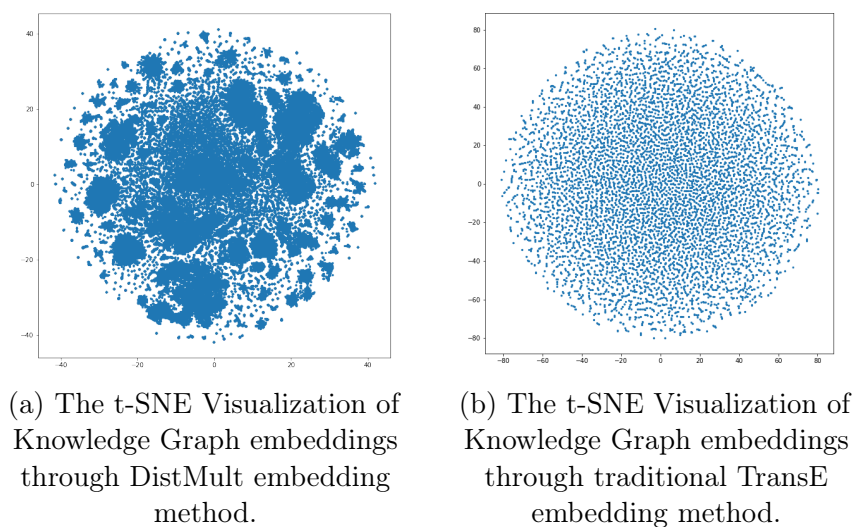


Figure 4.2: 2 Figures side by side

We preprocess the KG triples data by converting entities and relationships into embeddings. We use a pre-trained language model (**UMLSBERT [48]**) for encoding the given triples (h,r,t). We define a function (`convert_to_embeddings`) that takes a tokenizer, a model, and an input and returns the embedding representation of the input of the given KG. The function uses the encode method of the tokenizer to tokenize the input into a sequence of input IDs, which is then fed into the model to obtain the last hidden state of the model. The last hidden state is then used as the embedding representation. Finally, we generate embeddings for each entity and relationship in the triple data through the function. The embeddings are stored in the `entity_to_embedding` and `relation_to_embedding` dictionaries.

Node and Relation Embeddings and Explorations

Dense graph nodes refer to nodes with relatively high connections or edges compared to others in the same graph. Relation embedding is a technique used to represent relationships between nodes in a graph in a continuous, low-dimensional vector space called an embedding space. It can be helpful for node classification, link prediction, and visualization of the graph structure.

Exploration of dense space refers to analyzing and understanding the properties of the dense regions in the graph, such as identifying clusters of densely connected nodes or detecting patterns in the relationships between nodes. This can be done using graph clustering, community detection, and network centrality measures.

Evaluation Metrics

Evaluating the performance of KG (knowledge graph) embedding models typically involves using a combination of quantitative metrics and qualitative inspections.

Quantitative metrics that are commonly used to evaluate KG embedding performance include:

- **Mean Reciprocal Rank (MRR):** This metric measures the average rank of the correct entity in the list of entities returned by the embedding model. A higher MRR score indicates better performance.
- **Mean Rank (MR):** This metric measures the average rank of the correct entity across all test cases. A lower MR score indicates better performance.
- **Inverse geometric mean of ranks (IGMR):** This metric is like MRR but is a more robust performance measure. The IGMR can be computed as the reciprocal of the geometric mean of the ranks of the correct entities.

Qualitative inspections that are commonly used to evaluate KG embedding performance include:

- **Tail prediction:** This qualitative evaluation assesses the model’s ability to handle entities with fewer facts and less connectivity. The tail entities are considered the low-frequency entities in the dataset.

Evaluating the KG embedding performance requires a combination of quantitative metrics, such as **MR, MRR, and IGMR**, as well as qualitative inspections, such as **tail prediction**, to ensure that the model can handle entities with different levels of connectivity and facts.

The candidates are filtered to exclude triples seen in the train, validation, and test sets so that known triples do not affect the ranking and cause false negatives. Several ranking-based metrics are computed based on the sorted scores. Typical link prediction metrics include Mean Rank (MR), Mean Reciprocal Rank (MRR), and Hits@k (H@k). MR is sensitive to outliers and unreliable as a metric, using Mean Quantile (MQ) as a more robust alternative to MR and MRR. We use MQ100 as a more challenging version of MQ that introduces a cut-off at the top 100th ranking, appropriate for the large numbers of possible entities. Link prediction results are reported in Figure 4.3.

Knowledge Graph Link Prediction

| Model | MRR | MQ₁₀₀ | H@1 | H@10 |
|----------------------|------------|-------------------------|------------|-------------|
| TransE | .346 | .739 | .212 | .597 |
| ComplEx | .461 | .761 | .360 | .652 |
| DistMult | .420 | .752 | .309 | .626 |
| SimplE | .432 | .735 | .337 | .615 |
| RotatE | .317 | .742 | .162 | .599 |
| TransE _{FB} | .294 | - | - | .465 |
| TransE _{WN} | .226 | - | - | .501 |
| RotatE _{FB} | .338 | - | .241 | .533 |
| RotatE _{WN} | .476 | - | .428 | .571 |

Figure 4.3: Link prediction results: for the 5 KGE models on SNOMED-CT (top); and TransE and RotatE on two standard KGE datasets.

A standard evaluation task in the KGE literature is link prediction. It predicts whether the two nodes are connected or not. In addition, network Embeddings (NE) link prediction performs binary classification on a balanced set of positive and negative edges based on the assumption that the graph is complete. In contrast, knowledge graphs are typically assumed incomplete, making link prediction for KGE a ranking-based task in which the model’s scoring function is used to rank candidate samples without relying on ground truth negatives.

The link prediction refers to the latter ranking-based KGE method. Candidate samples are generated for each triple in the test set using all possible entities as the target entity, where the target can be set to head, tail, or both. For example, if the target is the tail, the model predicts scores for all possible candidates for the tail entity in (h, r, ?). The model calculates scores for fifteen billion candidate triples for a test set with 50k triples and 300k possible unique entities.

4.2 Baseline Model

BioLinkBERT [49] is a language model pre-trained on a large corpus of biomedical text data. It is based on the BERT architecture and has been fine-tuned on various biomedical NLP tasks, such as named entity recognition, relation extraction, and question answering. As a result, BioLinkBERT has achieved state-of-the-art performance on several benchmark datasets in the biomedical domain. We are Fine-tuning a pre-trained model BioLink-BERT on a medical dataset MedQA USMLE. As a result, we obtain a pre-trained

BioLink-BERT model. This can be done by downloading the model from the BioLink-BERT website or using a pre-trained version available in a deep learning framework such as TensorFlow or PyTorch.

```
tokenizer =
    AutoTokenizer.from_pretrained('michiyasunaga/BioLinkBERT-large')
model = AutoModel.from_pretrained('michiyasunaga/BioLinkBERT-large')
outputs = model(**inputs)
last_hidden_states = outputs.last_hidden_state
model = model.to("cuda")
```

List 4.1: Pre-trained Model

For the basic preprocessing, we import the AutoTokenizer, and AutoModel classes from the transformer's library. AutoTokenizer is used to tokenize text into a numerical representation that the model can process. The AutoModel class is used to instantiate the pre-trained model. In this case, the loaded model is BioLinkBERT-large, which the user "michiyasunaga" trained on the OpenAI model hub. The model is also moved to the GPU (if available) using the `.to("cuda")` method. We first use the tokenizer to convert the input texts into numerical representations that the model can process. The tokenized input is then moved to the GPU (if available) and passed as input to the pre-trained model specified by the model variable.

Variants of BioLinkBERT Models

There are several variants of BioLinkBERT models, and Hardware Configurations are following as:

1. **BioLinkBERT-Base:** a base variant trained on a sizeable biomedical corpus, the BioLinkBERT base size is trained on 110M parameters with the setting of learning rate $6e-4$, batch size 8,192, and training for 62,500 steps. We warm up the learning rate in the first 10% of steps and then linearly decay it. Training took seven days on eight A100 GPUs with fp16.

2. **BioLinkBERT-Large:** a larger version of BioLinkBERT that is trained on an even larger biomedical corpus containing the size of parameters 340M params. Follow the same procedure as -base, except use a peak learning rate of $4e-4$ and warm-up steps of 20%. Training took 21 days on eight A100 GPUs with fp16.

Hyperparameter Tuning

We fine-tune the pre-trained BioLink-BERT Large model on the medical dataset using transfer learning. It can be accomplished by using the pre-trained weights as the initial weights and training the model on the medical dataset using a task-specific objective, such as question answering or named entity recognition. The evaluation of the fine-tuned model on the validation set to determine the performance of the pre-trained model

Repeat the fine-tuning with different hyperparameter settings or architectures to find the optimal configuration. After Hyperparameter Tuning, we obtained the optimal weights, and we tested the model on the test set to evaluate its performance on unseen data and compare it with other models. It is worth noting that fine-tuning a BERT model, even with a specialized BioLink-BERT, requires a lot of computational resources and time, so we tested it on a powerful GPU or access cloud computing resources.

Set-up: Freebase and Virtuoso knowledge graph management

Freebase and Virtuoso are two examples of knowledge graph management systems that can be used for building and querying knowledge graphs. Freebase was a famous public knowledge graph developed by Google, which has since been deprecated. Virtuoso, on the other hand, is a commercial knowledge graph management system developed by OpenLink Software.

Freebase was a large-scale, collaborative knowledge graph developed by Metaweb Technologies and later acquired by Google in 2010. Freebase was a structured, open database of entities and their relationships, covering various topics and domains. The data in Freebase was contributed and maintained by a community of volunteers and could be accessed and queried using a query language called MQL (Metaweb Query Language). Freebase was designed to be a knowledge base for powering applications like search engines, recommendation systems, and question-answering systems. Therefore, the data in Freebase was structured in a way that made it easy to query and analyze using various tools and APIs.

Virtuoso is a high-performance, commercial knowledge graph management system developed by OpenLink Software. Virtuoso provides a platform for building and querying knowledge graphs and databases that store information about entities and their relationships in a structured format. Virtuoso supports a range of data models, including RDF, RDFS, OWL, and SPARQL, which are widely used in the semantic web community. Virtuoso is designed to support large-scale, distributed knowledge graphs that can be queried in real-time using

a variety of languages and APIs. In addition, it provides a range of features for managing data quality, versioning, access control, and data integration. Virtuoso is used by a range of organizations, including government agencies, research institutions, and enterprises, for building applications that require a semantic data layer, such as search engines, recommendation systems, and knowledge-based systems.

4.3 Data Collection Methods

For observing and analyzing the KG triples store, we organize the triplet store (triplet.tsv) in the two efficient data structures GraphDB and NetworkX, two different tools for collecting, storing and analyzing graph data. GraphDB is a native graph database that provides ACID transactions, a SPARQL 1.1 compliant query language, and reasoning support based on RDFS, OWL 2 RL, and OWL 2 EL. It is used for managing large-scale, complex and highly inter-connected data. It is designed to store and process large-scale graph data, making it an ideal choice for organizations with high performance and scalability requirements.

NetworkX, on the other hand, is a Python library for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks. It provides several algorithms for analyzing graph data, including shortest path algorithms, centrality measures, and community detection algorithms. NetworkX is used for data analysis and modelling but not for storing large-scale graph data. It is suited for small to medium-sized graphs and research and development purposes. GraphDB and NetworkX are useful for graph data analysis and have their strengths and weaknesses depending on the use case. The choice between the two will depend on the size and complexity of the graph data, the scalability requirements, and the specific use case.

4.3.1 Expressing Knowledge Graph Properties Through NetworkX

NetworkX is a Python library used to create, manipulate, and study the structure and dynamics and perform the complex functions of complex networks. It provides data structures for representing various types of networks (including graphs, directed graphs, and multigraphs) and algorithms for analyzing them. Some of the features of **NetworkX** include support for parallel edges and self-loops, node and edge attributes, and various graph generators. It also has several built-in drawing options for visualizing networks, as shown in Figure 4.4. **NetworkX** can be used for multiple tasks, such as studying social networks,

analyzing network traffic, and modeling biological systems.

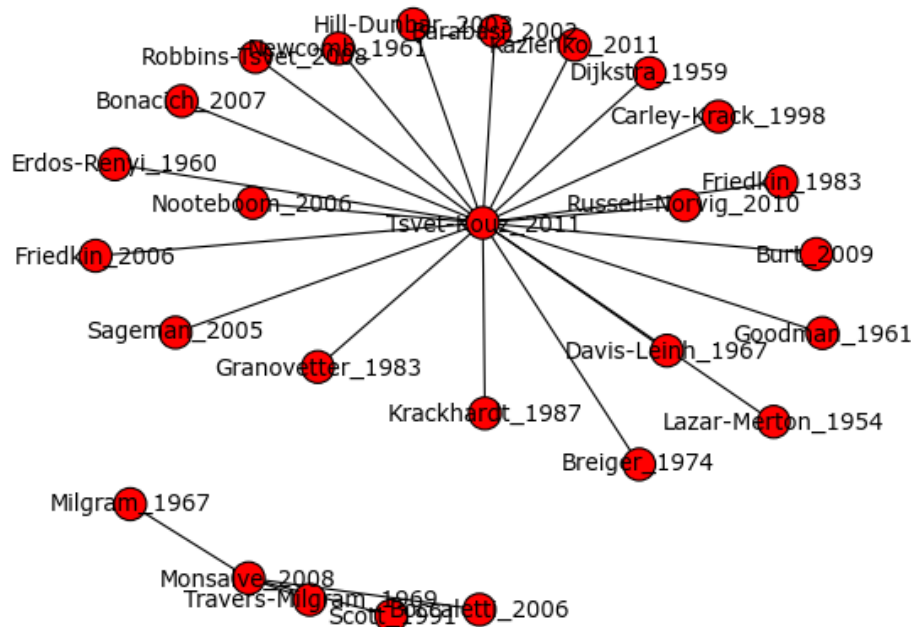


Figure 4.4: Directed graph with Networkx and Matplotlib.

The **Dragon Stanford KG** is a knowledge graph that contains a large amount of structured data about entities and their relations. NetworkX can be used to extract various properties from this knowledge graph, such as:

- **Average node degree:** The average degree of a node in a graph is calculated using the `nx.average_degree()` method.

$$average_degree = nx.average_degree(G)$$
- **Number of isolated nodes:** An isolated node is a node with no edges. The number of isolated nodes in a graph can be calculated using the `nx.isolates()` method.
 - $isolated_nodes = nx.isolates(G)$
 - $number_of_isolated_nodes = len(isolated_nodes)$
- **Inverse relations:** In a directed graph, an inverse relation is an edge that goes in the opposite direction of another edge. To find inverse relations in a directed graph, you can use the `in_edges()` and `out_edges()` methods

to get all incoming and outgoing edges for each node and then compare them.

inverse_relations = [(u, v) for u, v in G.in_edges() if (v, u) in G.out_edges()]

- **Number of connected components:** The number of connected components in a graph can be calculated using the *nx.number_connected_components()* method.

number_of_connected_components = nx.number_connected_components(G)

- **Degree centrality:** It measures the identifying importance of a node in a network based on the number of edges incident to it or connected. The degree centrality of a node can be calculated using the *nx.degree_centrality()* method.

There could be several reasons why a query execution fails in NetworkX on a USMLE dataset.

Some possible reasons include the following:

- **Incorrectly formatted data:** The USMLE dataset may be in a format different from NetworkX, or the data may be corrupted or missing.
- **Memory limitations:** The query may require more memory than is available, causing the execution to fail, and it requires more time for the execution of queries.
- **Inadequate computational resources:** The query may require more computational resources than are available, causing the execution to fail.

Limitation of directly querying on the graph using network X There are several limitations to directly querying a graph using NetworkX:

- **Complex queries can be challenging to express:** NetworkX is a general-purpose graph library. While it provides several built-in methods for querying graphs, defining more complex queries can be difficult and may require writing custom code.
- **Limited performance on large datasets:** NetworkX is designed for small to medium-sized graphs and may perform poorly on large datasets, especially for queries that require traversing the entire graph or a significant portion of it.

- **Limited scalability:** NetworkX is not built for distributed computing, so it may need to scale better to handle large datasets or high query loads.
- **Limited query language:** NetworkX does not provide a declarative query language like SQL or SPQRQL, making it more difficult for users unfamiliar with the library to express queries or understand the results.
- **Limited support for advanced graph analytics:** NetworkX provides a basic set of graph analysis functionalities, but it may not support advanced features such as graph partitioning, graph summarization, and machine learning on graph-structured data, which may need additional libraries or frameworks.

It's essential to consider the above limitations when using NetworkX for querying KG graphs and to use it accordingly.

4.3.2 GraphDB

Why GraphDB?

The first reason is that Google discontinued Freebase in 2016, and the Freebase data is no longer publicly available. As a result, the Freebase-Setup instructions are no longer applicable. While setting up the Virtuoso server can be complex and require technical expertise in database administration and semantic web technologies. So during setup, we faced issues and wrote to the author for a solution. However, we do not have access to Freebase, so we moved on to another Knowledge graph management called GraphDB and proceeded with it designing some queries from collecting data from the extractable triples.

GraphDB is a graph database management system (DBMS) that stores and manages graph data. A graph database is a database that uses a graph data model to store and manage data. In a graph database, data is stored as nodes and edges in a graph structure rather than in tables as in a traditional relational database. GraphDB is a knowledge graph management system that provides high performance, scalability, and flexibility for storing, querying, and managing large-scale knowledge graphs. It supports RDF (Resource Description Framework) and OWL (Web Ontology Language) data models, allowing it to be used for a wide range of knowledge representation and management tasks.

Some of the critical features of GraphDB include support for complex query patterns, reasoning, and inferencing over graph data, as well as integration with other tools and technologies, such as SPARQL, RDFS, and OWL. In addition, GraphDB provides advanced security and privacy features, such as

role-based access control, encrypted data storage, and privacy-preserving query evaluation.

Transformation of KG triples into RDF:

In GraphDB, onto text Refine is a web-based tool that can convert tabular data to RDF (Resource Description Framework) data as shown in Figure 4.5. The tool provides a simple and intuitive interface for cleaning, transforming, and modelling tabular data as RDF triples. We imported the pre-processed KG triples and the tabular data into Ontotext Refine. Ontotext Refine provides several options for exporting the RDF data, including RDF/XML, Turtle, N-Triples, and JSON-LD. We run the SPARQL queries in GraphDB that are executed against the RDF triples stored in the database and can be used to retrieve, filter, and aggregate data in the knowledge graph.

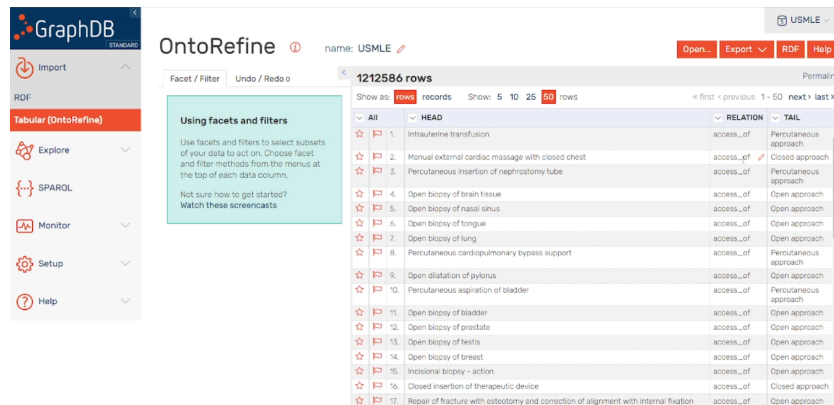


Figure 4.5: Data Transformation through OntoText Refine.

SPARQL queries in GraphDB can be used to perform a wide range of operations, including:

- Retrieving specific triples or data values based on specified conditions.
- Joining data across multiple triples or RDF graphs.
- Aggregating and summarizing data based on specified conditions.
- Performing complex reasoning and inferencing over RDF data.

```
def get_single_tail_relation_triplet(self, src):
    query = (f"""
        PREFIX rdf:
        <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
```



```

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT distinct ?r ?t0 WHERE {{
SERVICE <ontorefine:2567037650912> {{
    ?row a <http://rdf.freebase.com/ns/Row> ;
        <http://rdf.freebase.com/ns/HEAD> "{src}" ;
        <http://rdf.freebase.com/ns/RELATION> ?r ;
        <http://rdf.freebase.com/ns/TAIL> ?t .
}}
}}
""")
self.sparql.setQuery(query)
print(query)
try:
    results = self._load_from_disk(query)
    if not results:
        results = self.sparql.query().convert()
        self._save_to_disk(query, results)

```

List 4.2: Reteriving single tail relation triples

Searching for shortest paths, One hop path, two hops path

Query to retrieve relations between two entities (src and tgt) using a SPARQL endpoint. The query retrieves the particular r0 relation between src and tgt. If the query returns results, it returns the triplet (src, r0, tgt) as a list. If there are no results, the function `get_single_tail_relation_triplet` is called to get single-hop relations from src, and the code performs more operations to get two-hop relations.

The function `get_shortest_path_limit` method tries to find the shortest path between a source and a target entity from the Freebase knowledge graph using a SPARQL query. The method first queries for a direct relationship between the source and the target, and if there is, it returns a one-hop path. If not, it queries for the single tail relation triplets between the source, and for each relation, it queries for a relation between the relation's tail and the target. If there are results, it returns a two-hop path. The results of each query are saved to disk so they can be reused.

```

def search_one_hop_relaiotn(self, src: str, tgt: str) ->
List[Tuple[str]]:
    topic_entity = src
    answer_entity = tgt

```

```

query = f"""
    PREFIX rdf:
      <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
    PREFIX rdfs:
      <http://www.w3.org/2000/01/rdf-schema#>
    SELECT distinct ?r1 WHERE {{
      SERVICE <ontorefine:2567037650912> {{
        ?row a <http://rdf.freebase.com/ns/Row> ;
          <http://rdf.freebase.com/ns/HEAD>
            "{topic_entity}" ;
          <http://rdf.freebase.com/ns/RELATION>
            ?r0 ;
          <http://rdf.freebase.com/ns/TAIL>
            "{answer_entity}" .
      }}
    }}
"""

```

List 4.3: One Hop Relation Searching

```

def search_two_hop_relation(self, src, tgt) -> List[Tuple[str, str]]:
    topic_entity = src
    answer_entity = tgt
    query = f"""
        PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
        PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
        SELECT distinct ?r1 ?r2 WHERE {{
          SERVICE <ontorefine:2567037650912> {{
            ?row a <http://rdf.freebase.com/ns/Row> ;
              <http://rdf.freebase.com/ns/HEAD>
                "{topic_entity}" ;
              <http://rdf.freebase.com/ns/RELATION> ?r1 ;
              <http://rdf.freebase.com/ns/TAIL> ?e1 .

            ?row a <http://rdf.freebase.com/ns/Row> ;
              <http://rdf.freebase.com/ns/HEAD> ?e1 ;
              <http://rdf.freebase.com/ns/RELATION> ?r2 ;
              <http://rdf.freebase.com/ns/TAIL>
                "{answer_entity}" .
          }}
        }}
    """

```

```
"""  
  
# print(query)  
self.sparql.setQuery(query)  
print(query)
```

List 4.4: Two Hop Relation Searching

The give function `deduce_subgraph_by_path_one` function takes two parameters:

- **src:** a string representing the source node in the graph.
- **rels:** a list of strings representing the relationships in the graph.

The queries retrieve nodes connected to the `src` node via the first relationship in `rels`. The query results are the entities (nodes) connected to the `src` node through the first relationship.

The function then generates two outputs:

- **Nodes:** a list of nodes connected to the `src` node through the first relationship. It contains both the `src` node and the entities returned from the SPARQL query.
- **Triples:** a list of triples representing the relationships between the `src` node and the entities returned from the SPARQL query. The function returns the nodes and triples lists.

This `deduce_subgraph_by_path_two` defines a method `deduce_subgraph_by_path_two` that takes as input a source entity `src` and a list of relation strings `rels` and returns a list of nodes and a list of triples in the form of (subject, relation, object). The query that retrieves the distinct pairs of entities (`?e1`, `?e2`) that are connected by the two relationships `rels[0]` and `rels[1]` starting from the source entity `src`. The method then makes the query to the SPARQL endpoint, either by loading pre-saved results from the disk or making a new query if no saved results are found. Finally, the method returns the list of nodes and the list of triples, with duplicates removed.

4.4 Graph Merging

Basic Preprocessing and embeddings:

For the basic preprocessing, we import the `AutoTokenizer`, and `AutoModel` classes from the transformer's library `AutoTokenizer` is used to tokenize text

into a numerical representation that the model can process. The AutoModel class is used to instantiate the pre-trained model. In this case, the loaded model is BioLinkBERT-large, which was trained by the user "michiyasunaga" on the OpenAI model hub. The model is also moved to the GPU (if available) using the `.to("cuda")` method. We first use the tokenizer to convert the input texts into numerical representations that the model can process. The tokenized input is then moved to the GPU (if available) and passed as input to the pre-trained model specified by the model variable. Finally, the model function is called with the following parameters.

```
def get_texts_embeddings(texts):
    inputs = tokenizer(texts, padding=True,
                       truncation=True, return_tensors="pt")
    inputs = {k: v.to(device) for k, v in inputs.items()}
    embeddings = model(**inputs, output_hidden_states=True,
                      return_dict=True).pooler_output
    return embeddings

return list(candidate_relations)
```

List 4.5: Embeddings

Inferring path to subgraph

We need to input the `topic_entity` and their connected path for an inferring path to the subgraph. The topic entity is a node representing the starting node in a knowledge graph, and the path represents the edges following from the starting node to form a subgraph. Through this function, we got a set of nodes and a set of triples from the subgraph. To find the candidate relation from the given paths, we use the `topic_entity` and path and deduce the relations that can be formed between the leaf nodes of the subgraph specified by the path and the `topic_entity`. The function then filters out relations with a namespace of KG or common. Finally, the function returns the set of candidate relations as a list of strings.

```
def path_to_subgraph(topic_entity: str, path: List[str]):
    """Input topic_entity, path, and get the corresponding
       instantiated subgraph - node set, triple set"""
    return kg.deduce_subgraph_by_path(topic_entity, path)

def path_to_candidate_relations(topic_entity: str, path: List[str]) -
```

```

List[str]:
    """Enter topic_entity to get the set of candidate relation that
       leaf nodes can provide"""
    new_relations = kg.deduce_leaves_relation_by_path(topic_entity,
        path)
    # filter relation
    candidate_relations = [r for r in new_relations if
        r.split(".")[0] not in ["kg", "common"]]
    return list(candidate_relations)

```

List 4.6: Path to Candidate Relations

Scoring the path

The function takes in the input text (question), a list of paths (path_list), a list of scores for each path (path_score_list), a list of candidate relations for each path (relation_list_list), and a weighting factor (theta). First, for each path in path_list, the function concatenates the path and the question, separated by "#". Then, it calculates the embeddings for each query-path combination and the candidate relations. Finally, it calculates the cosine similarity between each query-path embedding and each candidate relation embedding, as shown in equation 3.4.

```

def score_path_list_and_relation_list(question: str, path_list:
    List[List[str]], path_score_list: List[float],
        relation_list_list:
            List[List[str]], theta: float =
            0.07) -> List[
    Tuple[List[str], float]]:
    """Calculate the score of the path and its corresponding
       candidate relation"""
    results = []

    query_lined_list = ['#'.join([question] + path) for path in
        path_list]m

    all_relation_list = list(set(sum(relation_list_list, [])))
    q_emb = get_texts_embeddings(query_lined_list).unsqueeze(1) # [B,
        1, D]
    target_emb = get_texts_embeddings(all_relation_list).unsqueeze(0)
        # [1, L, D]
    sim_score = torch.cosine_similarity(q_emb, target_emb, dim=2) /

```

```

theta # [B, L]
for i, (path, path_score, relation_list) in
    enumerate(zip(path_list, path_score_list, relation_list_list)):
    for relation in relation_list:
        j = all_relation_list.index(relation)
        new_path = path + [relation]
        score = float(sim_score[i, j]) + path_score
        results.append((new_path, score))
return results

```

List 4.7: Path Scoring

We scale the cosine similarity scores by dividing them by the value of theta. Finally, it adds each cosine similarity score to the corresponding path score for each path-relation combination and appends the resulting new path and score to a list of results.

Searching for the sequence of related paths

We aim to find relevant paths in a knowledge base (KB) or KG to answer a question. We start with an initial set of candidate paths, a list of paths with their corresponding scores. Then, at each iteration, we first calculate the next set of candidate relations for each path based on the current topic entity and the path history, as shown in equation 3.5. Then, we score each path-relation pair based on how well the pair matches the question and retains a certain number of the highest-scoring path-relation pairs as the new set of candidate paths. The process continues until either no more candidate paths are left or Until we Reach the virtual End keyword, as shown in equation 3.5, the maximum number of hops is reached, or the required number of result paths is found.

```

def _reverse_graph(G: Dict[str, List[str]]):
    r_G: Dict[str, List[str]] = dict()
    for u in G:
        for v in G[u]:
            r_G.setdefault(v, []).append(u)
    return r_G

```

List 4.8: Forward and Backward Tracing

```

def bfs_graph(G: Dict[str, List[str]], root):
    """
    G: an adjacency list in Dict

```

```

return: all bfs nodes
"""
visited = set()
currentLevel = [root]
while currentLevel:
    for v in currentLevel:
        visited.add(v)
    nextLevel = set()
    # levelGraph = {v:set() for v in currentLevel}
    for v in currentLevel:
        for w in G.get(v, []):
            if w not in visited:
                # levelGraph[v].add(w)
                nextLevel.add(w)
    # yield levelGraph
    currentLevel = nextLevel
return visited

```

List 4.9: Forward and Backward Tracing

In **graph searching**, we use two search strategies, Forward and backward tracing. In backward tracing, It takes a directed graph represented as a dictionary and returns a new graph with the same nodes but with reversed edges. While on the other forward tracing of the graph, it takes a graph and a root node and returns all nodes reachable from the root node using a breadth-first search.

Graph Merging

To merge two directed graphs, we have two graphs at the point graph-left and graph-right having two root-nodes root-l and rot-r, respectively. Therefore, the function asserts that root_l and root_r should be different. For merging, we start by finding the common nodes between the two graphs stored in the set common_nodes list. It then finds all the ancestors and descendants of each node in common nodes using the bfs_graph function on the reverse graph and the original graph of both graph_l and graph_r. The reverse graph of a directed graph is where all edges are reversed. Finally, it returns the set of all nodes, the union of the common nodes, ancestors, and descendants of the common nodes.

```

def merge_graph(graph_l, root_l, graph_r, root_r):
    assert root_l != root_r
    all_nodes = set()

```

```
common_nodes = set(graph_l) & set(graph_r)
all_nodes |= common_nodes # Union
reverse_graph_l, reverse_graph_r = _reverse_graph(graph_l),
    _reverse_graph(graph_r)
for node in common_nodes:
    ancestors_l = bfs_graph(reverse_graph_l, node)
    ancestors_r = bfs_graph(reverse_graph_r, node)
    descendants_l = bfs_graph(graph_l, node)
    descendants_r = bfs_graph(graph_r, node)
    all_nodes.update(ancestors_l)
    all_nodes.update(ancestors_r)
    all_nodes.update(descendants_l)
    all_nodes.update(descendants_r)
return all_nodes
```

List 4.10: Graph Merging

Chapter 5

Results

Query Execution Time

The execution time of a query on a database can depend on various factors, such as the query's complexity, like retrieving single-hop and multi-hop relations, the database's size, the resources available to the database, and the efficiency of the query execution engine. The given Figures show the query execution time and the number of operations needed during execution.

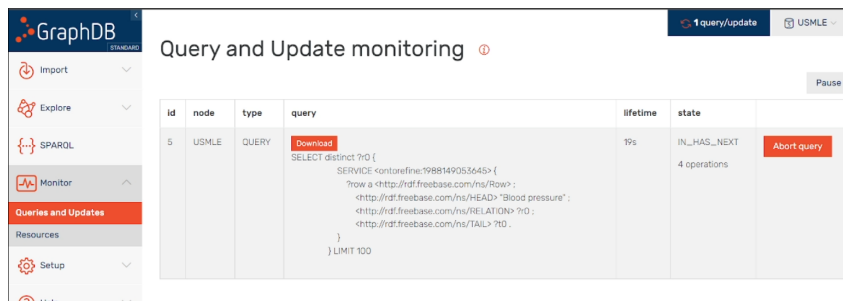
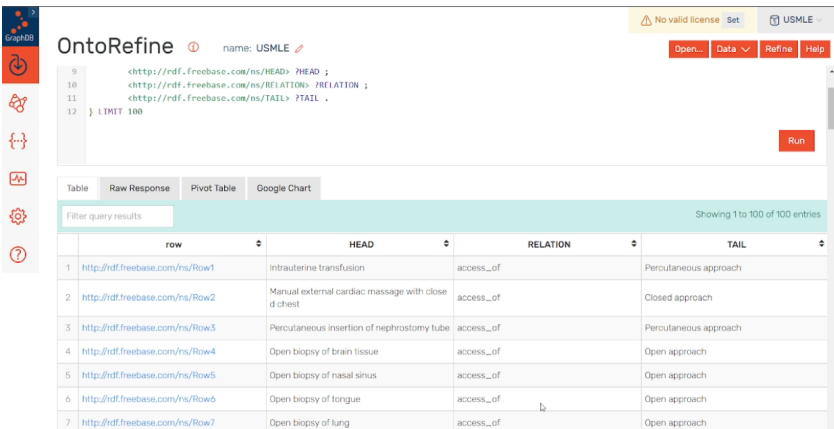


Figure 5.1: GraphDB: SPARQL query execution time for retrieving particular relations.

Retrieving Particular Relations and Caching Mechanism

Retrieving specific triples or data values based on specified conditions. For example, in the data collection phase, We use the SELECT operation to retrieve distinct values of the relation "?r0" compared to the given entities as shown in Figure 5.3. SERVICE clause to access data from an onto-refine server at the specified URL (localhost:8000) and retrieves rows that match certain conditions. The conditions are specified using a series of predicates, including the row type, head value, relation, and tail value.



The screenshot shows the OntoRefine web interface. At the top, there is a header with the name 'USMLE' and a 'Run' button. Below the header, a SPARQL query is displayed in a text area:

```

9 <http://rdf.freebase.com/ns/HEAD> ?HEAD ;
10 <http://rdf.freebase.com/ns/RELATION> ?RELATION ;
11 <http://rdf.freebase.com/ns/TAIL> ?TAIL .
12 } LIMIT 100

```

Below the query, there are tabs for 'Table', 'Raw Response', 'Pivot Table', and 'Google Chart'. The 'Table' tab is selected, and the results are displayed in a table with the following columns: 'row', 'HEAD', 'RELATION', and 'TAIL'. The table shows 7 rows of results, each representing a different medical procedure and its relationship to a specific approach.

| row | HEAD | RELATION | TAIL |
|-----|---|---|-----------------------|
| 1 | http://rdf.freebase.com/ns/Row1 | intrauterine transfusion | Percutaneous approach |
| 2 | http://rdf.freebase.com/ns/Row2 | Manual external cardiac massage with closed chest | Closed approach |
| 3 | http://rdf.freebase.com/ns/Row3 | Percutaneous insertion of nephrostomy tube | Percutaneous approach |
| 4 | http://rdf.freebase.com/ns/Row4 | Open biopsy of brain tissue | Open approach |
| 5 | http://rdf.freebase.com/ns/Row5 | Open biopsy of nasal sinus | Open approach |
| 6 | http://rdf.freebase.com/ns/Row6 | Open biopsy of tongue | Open approach |
| 7 | http://rdf.freebase.com/ns/Row7 | Open biopsy of lung | Open approach |

Figure 5.3: GraphDB: To execute a query on a graph database to retrieve relations from knowledge graph triples,

The existing approach represents the knowledge graph as sparse, where each node corresponds to an entity, and the edge corresponds to a relationship between entities. It is called discrete representation because the graph represents a set of nodes and edges. This type of knowledge graph can be done using SPARQL, a standard query language for RDF (Resource Description Framework) graphs. The strengths and weaknesses of sparse graph representation can be more intuitive and easier to reason about, but they can be slow to query for large graphs with complex patterns.

While the other efficient and fast approach is called dense vector representation, the dense vector representing knowledge graphs involves embedding the entities and relationships in a dense vector space. Querying this knowledge graph can be done using similarity search or other techniques designed for high-dimensional data. Dense representations can query and capture more complex relationships between entities faster, but they may need to be more transparent and easier to interpret.

We use suggested hyper-parameters from Dragon [9] for popular knowledge graph embedding models TransE and DistMult as follows:

For TransE:

TransE is a translation-based model where the embedding of the subject and object entities is translated by embedding the relation to predict the object entity. It is one of the earliest and most widely used models for knowledge graph embedding.

- learning rate: 0.01
- margin: 1.0

- L2 regularization weight: 0.0001
- dimension of embedding space: 200
- batch size: 1000
- number of negative samples per positive triple: 10
- number of training epochs: 100

For DistMult: DistMult is a bilinear model that models the interactions between the subject and object entities through the relation. It uses diagonal matrices to represent the relation embeddings, reducing the model’s number of parameters.

- learning rate: 0.1
- L2 regularization weight: 0.001
- dimension of embedding space: 100
- batch size: 512
- number of negative samples per positive triple: 1
- number of training epochs: 100

5.0.1 Knowledge Graph Embedding Models Performance Evaluation

Quantitative Performance Evaluation

The performance metrics (Hits@1, Hits@3, Hits@10) for the UMLS knowledge graph through KG embedding models such as TransE and DistMult.

Hits@1, Hits@3, and Hits@10 are evaluation metrics that measure the ability of a model to predict a true triple from a set of possible triples. Precisely, Hits@1 measures the proportion of test triples for which the true triple appears as the top-ranked prediction, Hits@3 measures the proportion of test triples for which the true triple appears within the top 3 predictions, and Hits@10 measures the proportion of test triples for which the true triple appears within the top 10 predictions as shown in Table 5.1.

Based on the provided performance metrics, Distmult outperforms then TransE, achieving the highest values for all three metrics. However, it is worth

| Model | Hits@1 | Hits@3 | Hits@10 |
|----------|--------|--------|---------|
| TransE | 0.46 | 0.58 | 0.71 |
| DistMult | 0.51 | 0.63 | 0.75 |

Table 5.1: Performance Metric on UMLS KG

noting that the choice of evaluation metric depends on the specific use case and goals of the knowledge graph embedding application.

Qualitative Performance Evaluation

DistMult

- **Head:** Intrauterine transfusion
- **Relation:** access_of

| Tail_id | Score | Tail |
|---------|-----------|---|
| 248602 | 0.964224 | Single photon emission computed tomography ... |
| 209380 | 0.954134 | Percutaneous approach |
| 134824 | 0.933178 | Hemoglobinuria |
| 114050 | 0.883393 | Fall from aircraft, not due to an accident to ... |
| 205534 | 0.864547 | Paralytic syndrome on both sides of the body |
| 189199 | -0.804073 | Nicotinamide adenine dinucleotide phosphate |
| 226464 | -0.858938 | Pupil normal |

Table 5.2: Tail Prediction using DistMult

The task of tail predictions for the given head entity "**Intrauterine transfusion**" and relation "**access_of**" using the **DistMult** model, the top predicted tails are taken from Table 5.2:

Tail 248602 with a score of 0.964224

Tail 209380 with a score of 0.954134

Tail 134824 with a score of 0.933178

Tail 114050 with a score of 0.883393

Tail 205534 with a score of 0.864547

The predicted tails suggest that the most likely entities to complete the triple are related to medical procedures or conditions, such as "**Single photon emission computed tomography,**" "**Percutaneous approach,**" and

"Hemoglobinuria." "access_of" in this context may be related to the accessibility of these medical procedures or conditions to the head entity "**Intrauterine transfusion.**" However, it is worth noting that knowing the specifics of the knowledge graph and the context in which the model is being used is necessary for further interpretation and analysis. Qualitative evaluation of model performance requires careful consideration of the specific use case and the domain knowledge surrounding the entities and relations in the knowledge graph.

TransE

| Tail_id | Score | Tail |
|---------|------------|--|
| 209380 | 0.761214 | Percutaneous approach |
| 134824 | 0.552414 | Hemoglobinuria |
| 58290 | -7.498745 | Closed approach |
| 131821 | -5.864547 | Closed reduction of fracture of femur and inte ... |
| 118086 | -14.804073 | HLA-Dw20 antigen |

Table 5.3: Tail Prediction using TransE

The given head entity "**Intrauterine transfusion**" and relation "**access_of**" using the **TransE** model, the top predicted tail through TransE embedding models as shown in Table 5.3:

Tail 209380 with a score of 0.761214

This tail score indicates that the TransE model is more confident in predicting the tail entity "Percutaneous approach" than any other entity given the head entity and relation.

The predicted tail suggests that the most likely entity to complete the triple is the "Percutaneous approach," a medical procedure for accessing or delivering something into the body through the skin, such as medication or a medical device. "access_of" in this context may be related to the accessibility of this medical procedure to the head entity "Intrauterine transfusion."

However, the low scores for the other tail predictions suggest that the TransE model may not be well-suited for this specific task. Qualitative evaluation of model performance requires careful consideration of the specific use case and the domain knowledge surrounding the entities and relations in the knowledge graph. Experimenting with different models or parameter settings may be necessary for better performance.

KG Embedding Comparison (TransE, DistMult, and CODER)

Knowledge graph embeddings are a powerful tool for representing the entities and relationships in a knowledge graph as dense vectors in a low-dimensional space. Doing so makes it possible to perform various tasks, such as link prediction, entity classification, and entity retrieval. Traditional embedding models, such as TransE and DistMult, have been widely used for this purpose, and they have demonstrated good performance on many benchmark datasets. However, their limitations include the inability to capture complex relationships between entities and the need for interpretability.

CODER, on the other hand, is a more recent embedding model specifically designed to address some of these limitations. It is based on representing the knowledge graph as a sequence of operations, which are then used to generate the embeddings. By doing so, CODER can capture more complex relationships between entities and is more interpretable than traditional embedding models.

In addition to its advantages for link prediction and entity classification, CODER can be used for term normalization, terms classification, and finding semantically similar terms from the space. Furthermore, CODER can capture high semantic relationships between entities and generate embeddings sensitive to subtle differences in meaning. Overall, CODER is a promising approach to knowledge graph embedding that has the potential to outperform traditional models in many applications. However, its effectiveness will ultimately depend on the specific characteristics of the knowledge graph and the task at hand. These quantitative and qualitative evaluation methods and results express which embedding models perform well on our medical UMLS data.

5.0.2 Training a CODER model

Training Parameters

BioBERT is a pre-trained language model designed explicitly for biomedical text-mining tasks. "**biobert_v1.1_pubmed**" refers to a version of BioBERT pre-trained on PubMed abstracts and PMC full-text articles. We train the `biobert_v1.1_pubmed` model on our UMLS triple data. We use UMLS Biomedical Knowledge Graph not only because the researcher from Stanford is working on it, but it is also a huge research topic, and Stanford uses only the sample of UMLS, but in general UMLS dataset has many data from different medical fields. Therefore, we use the lightweight size of UMLS for our experiments.

Knowledge-infused cross-lingual medical term embedding for term normalization used the following training parameters for their model:

- Learning rate: 0.001
- Batch size: 256
- Epochs: 300
- Dropout rate: 0.2
- Hidden layer size: 300
- Number of negative samples: 5
- Window size: 5
- Subsample threshold: 1e-5

These parameters were used to train the CODER model on the UMLS dataset, which contains biomedical data from different domains. In addition, the model was evaluated on the task of term normalization, which involves mapping variant expressions of medical terms to their standard forms. The results showed that the CODER model outperformed several baseline methods and achieved state-of-the-art performance on this task. We have the license of UMLS, the original size of UMLS is 5GB. However, we use the sample of UMLS data, as compared to our training epochs which are fewer than the original work of the coder because of the size of the data 5GB.

Training Loss

The training loss of the model is approximately zero after the training of 9000 iterations with the given hyper-parameters (`trans_margin = 1.0`, `train_batch_size = 32`), as shown in Figure 5.4.

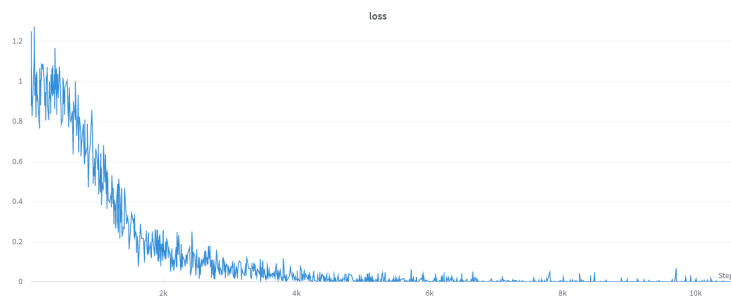


Figure 5.4: Coder: Training Loss

Hyperparameters such as margin and batch size can be used to control a model's training. The margin is a hyperparameter used in some loss functions,

such as the triplet or contrastive loss, commonly used in metric learning tasks. The margin represents the minimum distance or gap that must be maintained between different classes or examples in the embedding space. A larger margin can help ensure that embeddings of different classes are well-separated, while a smaller margin can allow for more overlap between classes.

Batch size is another hyperparameter that can control a model's training. It represents the number of examples processed together in each iteration of the training process. A larger batch size can lead to faster training times and more stable gradients but may require more memory and can lead to overfitting if the model is too complex. Conversely, the smaller batch size can lead to slower training times but can help prevent overfitting and may be necessary for models with limited memory.

Instead of PubMedBERT used for training the coder model, we use Biolinkbert to train CODER for tail prediction. So, for example, if we need to predict the tail of a sentence about a particular disease. First, we use transfer learning to fine-tune Biolinkbert on our UMLS dataset (Triple.csv). Adjusting the pre-trained weights of the model to fit better the specific text data you are working with. Once Biolinkbert has been fine-tuned, we use it for tail prediction and the training loss, as shown in Figure 5.4.

Conclusion and Future Work

Subgraph retrieval can be a complex and challenging task in biomedical question-answering systems. Biomedical datasets are often large, heterogeneous, and complex, with many different entity types, relationships, and attributes. Moreover, biomedical datasets may contain noisy or incomplete data, making retrieving accurate subgraphs difficult. For example, in a biomedical question-answering system, a user might ask, "What are the symptoms of a particular disease?" The system would need to retrieve a subgraph of relevant entities and relationships from the knowledge graph, such as the disease, its symptoms, and any associated treatments or risk factors. This would require understanding the semantics of the query, identifying the relevant entities and relationships in the graph, and ranking them based on relevance and accuracy. The above challenges and limitations can be solved through our proposed subgraph retrievals that can avoid entity linking tools instead of retrieving discrete or sparse graphs. In our contribution to UMLS KG, we focused on powerful embedding methods that are significantly faster and more efficient than the existing ones. Dense space embedding methods are faster and more efficient than the existing methods like the graph DB or SPARQL querying methods. For example, in the dense space setting, our search starts from the topic entities that are our starting point in the space. Our goal is to find or search the possible related relationships that help us reach the point in the space close to the answer or the path to the answer. We started from the topic entities and tried to predict the path more relevant to the starting entities in the space. Explored the space, measured the distance between the topic entities, and answered through the following paths. Moreover, it is how much the answer is far from the topic entities that how much reasoning hops require o inferring the answer to the given question. We score them according to their similarity score between the topic entities and the relation for the selection of paths, as mentioned in the section on subgraph constructions, and take the union of the produced trees and merge them into a standard unified graph.

Bibliography

- [1] Alexander Dunn, John Dagdelen, Nicholas Walker, Sanghoon Lee, Andrew S. Rosen, Gerbrand Ceder, Kristin Persson, and Anubhav Jain. Structured information extraction from complex scientific text with fine-tuned large language models, 2022.
- [2] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008, 2017.
- [3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics, 2019.
- [4] Renqian Luo, Liai Sun, Yingce Xia, Tao Qin, Sheng Zhang, Hoifung Poon, and Tie-Yan Liu. Biogpt: Generative pre-trained transformer for biomedical text generation and mining. *CoRR*, abs/2210.10341, 2022.
- [5] Zijun Yao, Yifan Sun, Weicong Ding, Nikhil Rao, and Hui Xiong. Dynamic word embeddings for evolving semantic discovery. In Yi Chang, Chengxiang Zhai, Yan Liu, and Yoelle Maarek, editors, *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining, WSDM 2018, Marina Del Rey, CA, USA, February 5-9, 2018*, pages 673–681. ACM, 2018.

-
- [6] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: online learning of social representations. In Sofus A. Macskassy, Claudia Perlich, Jure Leskovec, Wei Wang, and Rayid Ghani, editors, *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014*, pages 701–710. ACM, 2014.
- [7] Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. Knowledge graph embedding by translating on hyperplanes. In Carla E. Brodley and Peter Stone, editors, *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada*, pages 1112–1119. AAAI Press, 2014.
- [8] Xikun Zhang, Antoine Bosselut, Michihiro Yasunaga, Hongyu Ren, Percy Liang, Christopher D. Manning, and Jure Leskovec. Greaselm: Graph reasoning enhanced language models. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022.
- [9] Michihiro Yasunaga, Antoine Bosselut, Hongyu Ren, Xikun Zhang, Christopher D. Manning, Percy Liang, and Jure Leskovec. Deep bidirectional language-knowledge graph pretraining. *CoRR*, abs/2210.09338, 2022.
- [10] Jinhao Jiang, Kun Zhou, Wayne Xin Zhao, and Ji-Rong Wen. Unikgqa: Unified retrieval and reasoning for solving multi-hop question answering over knowledge graph. *CoRR*, abs/2212.00959, 2022.
- [11] Zheng Yuan, Zhengyun Zhao, Haixia Sun, Jiao Li, Fei Wang, and Sheng Yu. CODER: knowledge-infused cross-lingual medical term embedding for term normalization. *J. Biomed. Informatics*, 126:103983, 2022.
- [12] Jing Zhang, Xiaokang Zhang, Jifan Yu, Jian Tang, Jie Tang, Cuiping Li, and Hong Chen. Subgraph retrieval enhanced model for multi-hop knowledge base question answering. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio, editors, *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*, pages 5773–5784. Association for Computational Linguistics, 2022.
- [13] Jing Zhang, Xiaokang Zhang, Jifan Yu, Jian Tang, Jie Tang, Cuiping Li, and Hong Chen. Subgraph retrieval enhanced model for multi-hop knowledge base question answering. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio, editors, *Proceedings of the 60th Annual Meeting of*

- the Association for Computational Linguistics (Volume 1: Long Papers)*, *ACL 2022, Dublin, Ireland, May 22-27, 2022*, pages 5773–5784. Association for Computational Linguistics, 2022.
- [14] Michihiro Yasunaga, Hongyu Ren, Antoine Bosselut, Percy Liang, and Jure Leskovec. QA-GNN: reasoning with language models and knowledge graphs for question answering. In Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tür, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou, editors, *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2021, Online, June 6-11, 2021*, pages 535–546. Association for Computational Linguistics, 2021.
- [15] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In Zoubin Ghahramani, Max Welling, Corinna Cortes, Neil D. Lawrence, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 3104–3112, 2014.
- [16] Zachary Chase Lipton. A critical review of recurrent neural networks for sequence learning. *CoRR*, abs/1506.00019, 2015.
- [17]
- [18] Tomás Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In Christopher J. C. Burges, Léon Bottou, Zoubin Ghahramani, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*, pages 3111–3119, 2013.
- [19] Dongqin Xu, Junhui Li, Muhua Zhu, Min Zhang, and Guodong Zhou. Improving AMR parsing with sequence-to-sequence pre-training. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu, editors, *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*, pages 2501–2511. Association for Computational Linguistics, 2020.
- [20] Richard K. Lomotey and Ralph Deters. Real-time effective framework for unstructured data mining. In *12th IEEE International Conference on Trust*,

- Security and Privacy in Computing and Communications, TrustCom 2013 / 11th IEEE International Symposium on Parallel and Distributed Processing with Applications, ISPA-13 / 12th IEEE International Conference on Ubiquitous Computing and Communications, IUCC-2013, Melbourne, Australia, July 16-18, 2013*, pages 1081–1088. IEEE Computer Society, 2013.
- [21] Satoshi Sekine David Nadeau. A survey of named entity recognition and classification, 2007.
- [22] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692, 2019.
- [23] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.
- [24] Harsh Kumar, Ilya Musabirov, Jiakai Shi, Adele Lauzon, Kwan Kiu Choy, Ofek Gross, Dana Kulzhabayeva, and Joseph Jay Williams. Exploring the design of prompts for applying GPT-3 based chatbots: A mental wellbeing case study on mechanical turk. *CoRR*, abs/2209.11344, 2022.
- [25] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In Alessandro Moschitti, Bo Pang, and Walter Daelemans, editors, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1532–1543. ACL, 2014.
- [26] Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick S. H. Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu, editors, *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*, pages 6769–6781. Association for Computational Linguistics, 2020.
- [27] Bernardo Cuteri. Closed domain question answering for cultural heritage. In Viviana Mascardi and Ilaria Torre, editors, *Proceedings of the Doctoral Consortium of AI*IA 2016 co-located with the 15th International Conference of the Italian Association for Artificial Intelligence (AI*IA 2016), Genova, Italy, November 29, 2016*, volume 1769 of *CEUR Workshop Proceedings*, pages 17–22. CEUR-WS.org, 2016.

-
- [28] Quan Wang, Zhendong Mao, Bin Wang, and Li Guo. Knowledge graph embedding: A survey of approaches and applications. *IEEE Transactions on Knowledge and Data Engineering*, 29(12):2724–2743, 2017.
- [29] Heiko Paulheim. Knowledge graph refinement: A survey of approaches and evaluation methods. *Semantic Web*, 8(3):489–508, 2017.
- [30] Adrian Bondy. Graph-theory. online, 03 2008. Graph-Theory.
- [31] S. Gill Williamson Edward A. Bender. Lists, decisions and graphs - with an introduction to probability. online, 03 2008.
- [32] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In Yoshua Bengio and Yann LeCun, editors, *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*, 2013.
- [33] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In Balaji Krishnapuram, Mohak Shah, Alexander J. Smola, Charu C. Aggarwal, Dou Shen, and Rajeev Rastogi, editors, *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pages 855–864. ACM, 2016.
- [34] Andrea Rossi, Denilson Barbosa, Donatella Firmani, Antonio Matinata, and Paolo Merialdo. Knowledge graph embedding for link prediction: A comparative analysis. *ACM Trans. Knowl. Discov. Data*, 15(2):14:1–14:49, 2021.
- [35] Natasha Noy, Yuqing Gao, Anshu Jain, Anant Narayanan, Alan Patterson, and Jamie Taylor. Industry-scale knowledge graphs: Lessons and challenges. *Communications of the ACM*, 62 (8):36–43, 2019.
- [36] Soren Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary G. Ives. Dbpedia: A nucleus for a web of open data. In Karl Aberer, Key-Sun Choi, Natasha Fridman Noy, Dean Allemang, Kyung-Il Lee, Lyndon J. B. Nixon, Jennifer Golbeck, Peter Mika, Diana Maynard, Riichiro Mizoguchi, Guus Schreiber, and Philippe Cudre-Mauroux, editors, *The Semantic Web, 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007, Busan, Korea, November 11-15, 2007*, volume 4825 of *Lecture Notes in Computer Science*, pages 722–735. Springer, 2007.

- [37] Michihiro Yasunaga, Jure Leskovec, and Percy Liang. Linkbert: Pretraining language models with document links. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio, editors, *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*, pages 8003–8016. Association for Computational Linguistics, 2022.
- [38] Robyn Speer, Joshua Chin, and Catherine Havasi. Conceptnet 5.5: An open multilingual graph of general knowledge. In Satinder Singh and Shaul Markovitch, editors, *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*, pages 4444–4451. AAAI Press, 2017.
- [39] Niel Chah. OK google, what is your ontology? or: Exploring freebase classification to understand google’s knowledge graph. *CoRR*, abs/1805.03885, 2018.
- [40] Fangyu Liu, Ehsan Shareghi, Zaiqiao Meng, Marco Basaldella, and Nigel Collier. Self-alignment pretraining for biomedical entity representations. In Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tür, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou, editors, *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2021, Online, June 6-11, 2021*, pages 4228–4238. Association for Computational Linguistics, 2021.
- [41] Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. Convolutional 2d knowledge graph embeddings. In Sheila A. McIlraith and Kilian Q. Weinberger, editors, *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th Innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 1811–1818. AAAI Press, 2018.
- [42] Ronald Cornet and Nicolette de Keizer. Forty years of SNOMED: a literature review. *BMC Medical Informatics Decis. Mak.*, 8(S-1):S2, 2008.
- [43] Fangyu Liu, Ehsan Shareghi, Zaiqiao Meng, Marco Basaldella, and Nigel Collier. Self-alignment pretraining for biomedical entity representations. In Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tür, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou, editors, *Proceedings of the 2021 Conference of the*

- North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2021, Online, June 6-11, 2021*, pages 4228–4238. Association for Computational Linguistics, 2021.
- [44] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics, 2019.
- [45] Yu Gu, Robert Tinn, Hao Cheng, Michael Lucas, Naoto Usuyama, Xiaodong Liu, Tristan Naumann, Jianfeng Gao, and Hoifung Poon. Domain-specific language model pretraining for biomedical natural language processing, 2020.
- [46] Chen Zheng and Parisa Kordjamshidi. Dynamic relevance graph network for knowledge-aware question answering. In Nicoletta Calzolari, ChuRen Huang, Hansaem Kim, James Pustejovsky, Leo Wanner, Key-Sun Choi, Pum-Mo Ryu, Hsin-Hsi Chen, Lucia Donatelli, Heng Ji, Sadao Kurohashi, Patrizia Paggio, Nianwen Xue, Seokhwan Kim, Younggyun Hahm, Zhong He, Tony Kyungil Lee, Enrico Santus, Francis Bond, and Seung-Hoon Na, editors, *Proceedings of the 29th International Conference on Computational Linguistics, COLING 2022, Gyeongju, Republic of Korea, October 12-17, 2022*, pages 1357–1366. International Committee on Computational Linguistics, 2022.
- [47] Rajarshi Das, Ameya Godbole, Ankita Naik, Elliot Tower, Manzil Zaheer, Hannaneh Hajishirzi, Robin Jia, and Andrew McCallum. Knowledge base question answering by case-based reasoning over subgraphs. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvári, Gang Niu, and Sivan Sabato, editors, *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pages 4777–4793. PMLR, 2022.
- [48] Zheng Yuan, Zhengyun Zhao, Haixia Sun, Jiao Li, Fei Wang, and Sheng Yu. CODER: knowledge-infused cross-lingual medical term embedding for term normalization. *J. Biomed. Informatics*, 126:103983, 2022.
- [49] Michihiro Yasunaga, Jure Leskovec, and Percy Liang. Linkbert: Pretraining language models with document links. In *Association for Computational Linguistics (ACL)*, 2022.