Leveraging Textual Semantics for Knowledge Graph Acquisition and Application

Donghan Yu

CMU-LTI-23-010

Language Technology Institute
School of Computer Science
Carnegie Mellon University
5000 Forbes Ave., Pittsburgh, PA 15213
www.lti.cs.cmu.edu

Thesis Committee:
Yiming Yang (Chair)
Emma Strubell
Chenyan Xiong
Chenguang Zhu (Microsoft)

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
In Language and Information Technologies

© 2023, Donghan Yu

Keywords: knowledge graph, textual semantics, pre-trained language model, graph neural network, question answering

For my grandfather

Abstract

Knowledge Graphs (KGs), which represent world knowledge through entities and relations, are ubiquitous in real-world applications. Besides their structural nature, KGs offer rich textual information, as entities usually correspond to real-world objects with specific names and descriptions. Despite the importance of such information, it has been largely overlooked or inadequately explored in existing studies.

This thesis aims to integrate the textual information into the modeling of KGs by utilizing Pre-trained Language Models (PLMs), which have demonstrated effectiveness in capturing the semantic meanings of natural language. This goal is carried out on two complementary parts: the acquisition of KGs to enhance their qualities, and the application of KGs to address user queries.

In Part I, we focus on KG acquisition through text. We begin with a pre-training framework that jointly learns the vector representations of KGs and text. It features KG-text dual modules that mutually enhance each other, achieving strong results on relation extraction and entity classification. (Chapter 2). To address scalability challenges in large KGs, we propose a retrieval-enhanced text-generation model for KG completion. It leverages semantically relevant triplets from KGs to guide the generation of missing entities, demonstrating state-of-the-art performance while maintaining low memory usage (Chapter 3).

In Part II, we turn our attention to applying KGs to the crucial task of Question Answering (QA). In the setting that the answers are sourced from KGs, we propose a framework that jointly generates logical queries and text answers to produce more accurate and robust results (Chapter 4). Then we extend to the scenarios where the answers mainly stem from text corpora instead of KGs. Our proposed method leverages KGs to construct links among the text passages. Such structural information is leveraged to re-rank and prune related passages for each question, significantly reducing computational costs (Chapter 5). Finally, we tackle the setting of incomplete KGs. We introduce the first benchmark dataset to assess the impact of KG completion methods on question answering. Our experiments highlight the necessity to jointly study the acquisition and application of KGs (Chapter 6).

Acknowledgments

Pursuing a PhD is difficult, and I've been fortunate to receive support in both academic and personal aspects, giving me the strength to finish the journey.

First and foremost, I would like to express my gratitude to my PhD advisor, Yiming Yang, who has taught me the art of critical thinking and the skill of academic writing. She stresses the importance of advocating for what we genuinely believe is significant, rather than merely what is currently trending in the field. This thesis would not be possible without her guidance and support.

I would also like to extend my appreciation to my thesis committee members– Chenguang Zhu, Chenyan Xiong, and Emma Strubell–for their valuable feedback. I am particularly grateful to Chenguang, who was also my internship mentor at Microsoft. He showed me what qualities make a good team manager and how to lead a team of research scientists to achieve significant results in the industry.

I want to thank my group members and collaborators: Yuexin Wu, Zihang Dai, Ruochen Xu, Wei-Cheng Chang, Guokun Lai, Jingzhou Liu, Ruohong Zhang, Zhiqing Sun, Aman Madaan, Yuwei Wu, Shanda Li, Shengyu Feng, Zhengbao Jiang, Yuwei Fang, Wenhao Yu, Shuohang Wang, Yichong Xu, Sheng Zhang, Patrick Ng, Henghui Zhu, Alexander Hanbo Li, Jun Wang, Yiqun Hu, William Wang, Zhiguo Wang, Xiang Ren, and many others not listed here. Working with you all has been a truly enjoyable experience.

My life in Pittsburgh has been greatly enriched thanks to my friends. In particular, Jacob Arnold and Kelly Arnold stand out as my best friends in the US, and the friends I've spent the most time with during my PhD. I feel fortunate to have found such compatible and supportive friends in a foreign country. Special mention must also go to Wei Dong and Chunyu Shen. Our get-togethers are consistently filled with laughter and joy. Their delicious cooking is one of the things that I'll truly miss. In addition, I want to thank my little furry friend, Plus. Every time I see her, any negative moods fade away, leaving me filled with happiness.

I'd like to express my deepest gratitude to my parents for teaching me the most essential lesson in life, something even the most advanced machines fail to understand: the concept of love. This thesis is dedicated to my grandfather. His guidance and care for me are the memories that I will cherish forever. I wish he would be proud of what I've accomplished.

Last but not least, special thanks to my wife, Jessica Lee. For more than five years, she has been by my side, supporting me through many important moments in my life. I'm fortunate to have her in the past, grateful to be with her in the present, and I look forward to sharing life with her in the future.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Objects in the real world don't exist in isolation; they share semantic connections with one another. For example, a medicine can be related to a disease as its treatment, and a company can be linked to a video game as its developer. Knowledge Graphs (KGs) capture these connections, representing them as graph structures. In KGs, the objects are referred to as entities (nodes), while the connections between them are known as relations (edges). The fundamental unit within a KG is called a triplet, comprised of three elements: the head entity, relation, and tail entity, such as (*Elden Ring*, *developed_by*, *FromSoftware*), which conveys that the video game Elden Ring is developed by the game company FromSoftware. The graph structures of KGs allow them to illustrate various connections between real-world objects and facilitate complex queries and analyses. In addition, KGs focus on capturing essential connections over important objects and eliminating incorrect or irrelevant details. Given these benefits, KGs have been applied to various real-world scenarios such as question answering [118], information retrieval [64], and recommender systems [104], and they are implemented in different domains including finance [26], marketing [27], medicine [82], and more.

The significance of KGs has drawn widespread interest within the machine learning field. Many studies predominantly concentrate on the structure of KGs, while representing entities or relations as mere indices, thereby overlooking their textual information. For example, the entity *Elden Ring* is just represented as a random unique index like 1203 with no semantic meaning, ignoring its textual name "Elden Ring" and description "Elden Ring is a 2022 action role-playing game developed by FromSoftware...". This hinders both the acquisition and application of KGs.

To illustrate, consider the process of extracting triplets from text statements. When encountering a statement such as "FromSoftware received wide recognition for developing the video game Elden Ring", it's much easier to add or verify the text-enriched triplet ("Elden Ring", "developed by", "FromSoftware"), as opposed to a purely index-based triplet like $(1203, 47, 1580)$. Similarly, when applying KGs to assist with a user's text query, such as "Can I play Elden Ring on PS5?", having the text-enriched triplet ("Elden Ring", "platform", "PlayStation 5") eases the generation of an accurate answer. Without such textual information, aligning the text sentence or user query with mere entity and relation indices can become a complex obstacle. These examples highlight the essential role of textual information in KGs. It not only simplifies the acquisition of relational data but also enhances the functionality of KGs in real-world applications.

To effectively leverage such textual information, sematic-level understanding is necessary.

Consider again the query "Can I play Elden Ring on PS5?" To accurately respond to such a query, the model must recognize that "PS5" refers to the entity *PlayStation 5*, and the phrase "play...on..." corresponds to the relation *platform*. Then the model can locate the corresponding triplet (*Elden Ring*, *platform*, *PlayStation 5*) and provide the correct answer. This interpretation goes beyond mere word recognition; it demands an understanding of the underlying semantics of textual information. This kind of understanding is especially critical in real-world situations, where KGs often have formal and limited naming conventions, while user queries or statements can be informal and diverse.

1.1 Thesis Overview

This thesis aims to develop novel models for effectively utilizing textual semantics in both KG acquisition and application. It builds on the foundation laid by existing Pre-trained Language Models (PLMs) [9, 22, 77], which have shown dominance in textual semantics modeling. PLMs are pre-trained on large-scale corpora by predicting subsequent or masked words to learn representations of language. Leveraging deep learning architectures, such as transformer [97]-based models, they serve as powerful feature extractors and can easily adapt to various downstream tasks. However, since PLMs are designed for unstructured text sequences, their application to structured KGs presents a significant challenge. To address this challenge, our thesis is carried out through two important parts of concentration: 1) KG acquisition through text, which focuses on building more accurate and comprehensive KGs; 2) Answering text questions with KGs, targeting the application of KGs in resolving user queries.

1.1.1 KG Acquisition Through Text

The objective of KG acquisition is to enhance the quality of KGs through multiple tasks such as relation extraction and entity classification. The former focuses on adding or validating triplets based on textual evidence, whereas the latter aims to categorize entities into pre-defined labels. One effective technique for these tasks is to learn the vector representations of text sentences or KG entities. To achieve a synergistic improvement of both text and KG representations, we developed JAKET. It is a joint pre-training framework that features two modules designed to learn their specific representations while also mutually enhancing each other. On the one hand, the KG module creates structural embeddings for entities found in text, thereby enriching the text-based embeddings. On the other hand, the text module generates textual embeddings for KG entities and relations by using their names and descriptions. This dual approach enables a comprehensive modeling of structural and textual aspects within KGs. Our experiments reveal that JAKET offers superior performance in KG acquisition tasks like relation extraction and entity classification (Chapter 2).

Although effective, learning representation for all the entities can be memory-consuming, particularly when scaling to large KGs with millions of entities. This becomes a challenge in tasks like KG completion, which involves predicting missing entities in incomplete triplets. For scalability, recent research has framed KG completion as a Sequence-to-Sequence (Seq2Seq) problem, converting both the incomplete triplet and the missing entity into text sequences. While

various PLMs can assist in this transformation, inference with these methods relies solely on the model parameters for implicit reasoning while neglecting the use of KG itself. This limits their performance since the models usually lack the capacity to memorize a vast number of triplets. To tackle this issue, we introduce ResKGC, a Retrieval-enhanced Seq2Seq KGC model, which retrieves semantically relevant triplets from the KG and uses them as evidence to guide output generation via explicit reasoning. For example, given the incomplete triplet (*Hidetaka Miyazaki*, *work_at*, *?*), ResKGC could retrieve existing triplets like (*Elden Ring*, *director*, *Hidetaka Miyazaki*) and (*Elden Ring*, *developer*, *FromSoftware*) to assist in predicting the missing entity *FromSoftware*. ResKGC has demonstrated state-of-the-art performance on large-scale KGC benchmark datasets, while maintaining a lightweight parameter configuration (Chapter 3).

1.1.2 Answering Text Questions with KGs

In this part, we turn our attention to how KGs can be applied in question answering. Firstly, we focus on scenarios where the answers are restricted to the information available in KGs, commonly known as KGQA. Given a question such as "where is the company that won GOTY 2022 located?", previous methods either parse it into a logical query `(JOIN headquarter (JOIN developer (JOIN (R award) Q110028874))` that can be executed over KGs to obtain the final answers or predict the answer "Tokyo" directly without generating the intermediate queries. Empirical results show that the former often produces more accurate answers, but it suffers from non-execution issues due to potential syntactic and semantic errors in the generated logical queries. We propose a novel framework DecAF that jointly generates both logical queries and direct answers, which are produced by a reading comprehension module, and then combines them to get the final answers. In this case, if the logical queries are not executable, the direct answers can still serve as output (Chapter 4).

Then we go beyond to the task of Open-Domain Question Answering (ODQA), where the answers mainly locate in text corpora, instead of KGs. Current ODQA models typically include a retrieval module and a reading module: for a given question such as "Can I upgrade weapons in souls series?", the retriever selects potentially relevant text passages such as "In Dark Souls series, players can increase the levels of weapons by ..." from open-source documents, and the reader produces an answer "Yes" based on the retrieved passages. The Fusion-in-Decoder (FiD) framework [46] is a representative example, built on top of PLM-based retriever and reader, achieving state-of-the-art performance. Despite its effectiveness, the framework processes each passage independently, thereby overlooking the interrelationship among passages. In this study, we further improve the FiD approach by introducing a KG-enhanced version, namely KG-FiD. Our new model uses a knowledge graph to establish the structural relationships among the retrieved passages. For example, given the triplet (*Dark Souls Series*, *contain*, *Weapon*), the passages with title "Dark Souls Series" will be connected to the passages with title "Weapon". Such structural relationships are leveraged within both the retriever and reader to re-rank the passages and select only the top-ranking ones for subsequent processing (Chapter 5).

Finally, we aim to bridge KG acquisition and application by answering an essential but unexplored question: How much success in KG completion would translate into performance enhancement in KGQA? We introduce a novel benchmark, namely CompleQA, to comprehensively answer this question. This benchmark includes a knowledge graph with 3 million triplets

across 5 distinct domains, coupled with over 5000 question-answering pairs and a KGC dataset that is well-aligned with these questions. Our evaluation of four well-known KGC methods in combination with two state-of-the-art KGQA systems validates the common belief that KG completion can improve question-answering performance in most cases. On the other hand, surprisingly we also find that the best-performing KGC method(s) does not necessarily lead to the best question-answering results, owing to the complexity of PLM-based KGQA methods. Our findings highlight the significance of jointly studying KG acquisition and application under the context of PLMs, rather than investigating them in isolation (Chapter 6).

1.2 Preliminary

In this section, we introduce some mathematical notations and definitions related to KGs for easier interpretation of subsequent sections.

A knowledge graph can be represented by the notation $\mathcal{KG} = (\mathcal{E}, \mathcal{R}, \mathcal{T})$, where $\mathcal{E}$ is the set of entities, $\mathcal{R}$ is the set of relationships among entities, and $\mathcal{T}$ is the set of entity-relation-entity triplets which are semantically valid. A triplet can be denoted as (e_h, r, e_t), including head entity $e_h \in \mathcal{E}$, relation $r \in \mathcal{R}$, and tail entity $e_t \in \mathcal{E}$. $N_v = \{(r, u)|(v, r, u) \in \mathcal{T}\} \cup \{(u, r)|(u, r, v) \in \mathcal{T}\}$ represents the set of neighboring relations and entities of an entity v.

Representation learning for KGs, also known as knowledge graph embedding, aims to map the entities and relations into a continuous low-dimensional space while preserving the semantics inherent in the data, which is symbolized below:

$$f_{\text{embedding}} \colon \mathcal{E} \cup \mathcal{R} \rightarrow \mathbb{R}^{d_e} \cup \mathbb{R}^{d_r} \tag{1.1}$$

where d_e and d_r represent the dimensionality of the entity and relation embedding spaces, respectively. These learned representations can then be used in various downstream tasks covering both KG acquisition and application.

Knowledge Graph Completion (KGC), is the task of predicting missing triplets in KGs, and can be defined as the mapping from the initially incomplete KG to the extended (completed) KG:

$$f_{\text{KGC}} \colon \mathcal{KG} = \{\mathcal{E}, \mathcal{R}, \mathcal{T}\} \rightarrow \mathcal{KG}' = \{\mathcal{E}, \mathcal{R}, \mathcal{T}'\} \tag{1.2}$$

where $\mathcal{T}' = \mathcal{T} \cup \mathcal{T}_{\text{new}}$ constitutes the enriched triplet set containing original triplets and newly predicted triplets. To facilitate easier evaluation, many research studies focus on a surrogate task of KGC: missing entity prediction. In this task, given an incomplete triplet either in the form of $(e_h, r, ?)$ or $(?, r, e_t)$, the model is required to predict the missing entity e_t or e_h.

The application of KGs to Question Answering (QA) targets the interaction between KGs and natural language questions. This task, also known as KGQA, can be represented as follows:

$$f_{\text{QA}} \colon \mathcal{KG} \cap Q \rightarrow A \tag{1.3}$$

where Q represents the question, expressed by a text sequence, such as "Which game company developed the Dark Souls?", and A signifies the answers, which can be a text string like "FromSoftware" or a set of text strings. In a more general setting, the input can be $(\mathcal{KG} \cup \mathcal{D}) \cap Q$, where $\mathcal{D}$ refers to other data sources where answers may locate besides KG, such as text documents.

1.3 Related Publications

The KG-text joint pre-training framework in Chapter 2 was published at AAAI 2022 [123]. The retrieval-enhanced seq2seq KGC method in Chapter 3 was accepted to SIGIR 2023. The joint answer and logical form generation framework for KGQA in Chapter 4 was published at ICLR 2023 [124]. The KG-enhanced open-domain question answering method in Chapter 5 was published at ACL 2022 [122]. Finally, the benchmark study on the impact of KGC on KGQA in Chapter 6 is currently under review at a top conference.

The thesis also led to several other related works not listed as individual chapters, including a GNN framework that jointly learns edge prediction and node classification published at ECML-PKDD 2020 [121], the introduction of a new task of relation entailment prediction published at AKBC 2020 [48], a language model pre-training framework enhanced with dictionary information published at ACL 2022 [125], and an extreme multi-label text classification method leveraging dense neural retrieval and pseudo label generation published at EACL 2023 [127].

Part I

KG Acquisition Through Text

Chapter 2

Joint Pre-training of Knowledge Graph Embedding and Language Modeling

2.1 Introduction

Pre-training has demonstrated significant effectiveness in both text and KGs, although they excel in different aspects. In the context of text, pre-trained language models (PLMs) like BERT [22], T5 [77], and GPTs [9, 75, 76] employ transformer [97]-based architectures to learn token representations. Specifically, they learn by predicting masked or subsequent tokens and, as a result, generate high-quality, context-aware representations. Such models have achieved outstanding performance in a range of NLP tasks. However, a key limitation is their struggle to encode world knowledge about entities and relations effectively.

In the realm of KGs, pre-training focuses on learning both entity and relation representations. This is generally achieved through masked entity or relation prediction tasks. Methods for this include both non-neural network ones like TransE [6] and RotatE [91], as well as neural network-based ones such as ConvE [21] and RGCN [87]. These models excel at capturing the structural information present in KGs, but they often fall short when it comes to integrating textual information.

Recent efforts have aimed to bridge the gap between text and KG pre-training. These approaches generally fall into two categories. The first category enhances text-based pre-training by incorporating knowledge from KGs [28, 71, 100, 129]. The second category utilizes PLMs to improve the pre-training on KGs [107, 117]. However, these efforts are unidirectional in their enhancement strategy. They tend to prioritize either text or KGs, but not both, leading to a lack of mutual improvement between the two domains.

Therefore, we propose JAKET, a Joint pre-trAining framework for KnowledgE graph and Text. Our framework contains a knowledge module and a language module, which mutually assist each other by providing required information to achieve more effective semantic analysis. The knowledge module is based on a graph attention network [98] to provide structure-aware entity embeddings for language modeling. And the language module produces contextual representations as initial embeddings for KG entities and relations given their descriptive text. Thus, in both modules, content understanding is based on related knowledge and rich context. On one

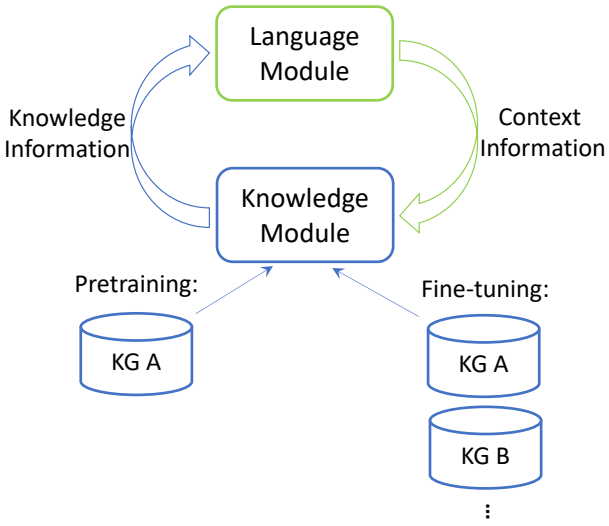


Figure 2.1: A simple illustration on the novelty of our proposed model JAKET: (1) The language module and knowledge module mutually assist each other; (2) JAKET can adapt to unseen knowledge graphs during fine-tuning.

hand, the joint pre-training effectively projects entities/relations and text into a shared semantic latent space, which eases the semantic matching between them. On the other hand, as the knowledge module produces representations from descriptive text, it solves the over-parameterization issue since entity embeddings are no longer part of the model's parameters.

In order to solve the cyclic dependency between the two modules, we propose a novel two-step language module LM_1 and LM_2, respectively. LM_1 provides embeddings for both LM_2 and KG. The entity embeddings from KG are also fed into LM_2, which produces the final representation. LM_1 and LM_2 can be easily established as the first several transformer layers and the rest layers of a pre-trained language model such as BERT and RoBERTa. Furthermore, we design an entity context embedding memory with periodic update which speeds up the pre-training by about 15x.

The pre-training tasks are all self-supervised, including entity category classification and relation prediction for the knowledge module, and masked token prediction and masked entity prediction for the language module.

A great benefit of our framework is that it can easily adapt to unseen knowledge graphs in the fine-tuning phase. As the initial embeddings of entities and relations come from their descriptive text, JAKET is not confined to any fixed KG. With the learned ability to integrate structural information during pre-training, the framework is extensible to novel knowledge graphs with previously unseen entities and relations, as illustrated in Figure 2.1.

We conduct empirical studies on KG acquisition tasks, including few-shot relation classification and entity classification. The results show that JAKET achieves the best performance compared with strong baseline methods on all the tasks.

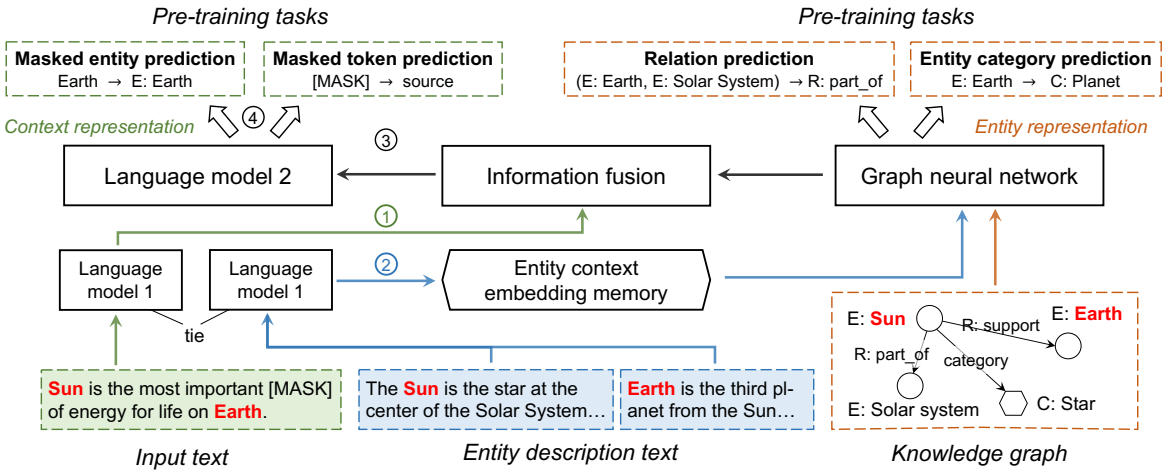


Figure 2.2: A demonstration for the structure of JAKET, where the language module is on the left side while the knowledge module is on the right side. Symbol ⓘ indicates the steps to compute context representations. "E:", "R:" and "C:" stand for Entities, Relations and Categories in KG respectively. Entity mentions in text are marked red and bold such as **Sun**.

2.2 Method

In this section, we introduce the JAKET framework of joint pre-training knowledge graph and language understanding. We begin by defining the mathematical notations, and then present our model architecture with the knowledge module and language module. Finally, we introduce how to pre-train our model and fine-tune it for downstream tasks. The framework is illustrated in Figure 2.2.

2.2.1 Definition

We define $\mathcal{V} = \{[\text{MASK}], [\text{CLS}], [\text{EOS}], w_1 \dots w_V\}$ as a vocabulary of tokens and the contextual text $\mathbf{x} = [x_1, x_2, \dots, x_L]$ as a sequence of tokens where $x_i \in \mathcal{V}$. In the vocabulary, [MASK] is the special token for masked language modeling [22] and [CLS], [EOS] are the special tokens indicating the beginning and end of the sequence. We define F as the dimension of token embeddings, which is equal to the dimension of entity/relation embeddings from the KG.

The text $\mathbf{x}$ has a list of entity mentions $\mathbf{m} = [m_1, \dots, m_M]$, where each mention $m_i = (e_{m_i}, s_{m_i}, o_{m_i})$: e_{m_i} is the corresponding entity and s_{m_i}, o_{m_i} are the start and end index of this mention in the context. In other words, $[x_{s_{m_i}}, \dots, x_{o_{m_i}}]$ is linked with entity e_{m_i}[1]. We assume the span of mentions are disjoint for a given text sequence.

As entities in the knowledge graph are represented by nodes without context, we use *entity description text* to describe the concept and meaning of entities. For each entity e_i, its description text $\mathbf{x}^{e_i}$ describes this entity. The mention of e_i in $\mathbf{x}^{e_i}$ is denoted as $m^{e_i} = (e_i, s_i^e, o_i^e)$, similarly defined as above. For instance, the description text for the entity "sun" can be "[CLS] The Sun

[1]We do not consider discontinous entity mentions in this work.

is the star at the center of the Solar System [EOS]". Then the mention is $m^{Sun} = (Sun, 3, 3)$. If there are multiple mentions of e_i in its description text, we choose the first one. If there's no mention of e_i in its description text, we set $s_i^e = o_i^e = 1$. Similarly, we define *relation description text* as the text that can describe each relation.

2.2.2 Knowledge Module

The goal of the knowledge module (KM) is to model the knowledge graph to generate knowledge-based entity representations.

To compute entity node embeddings, we employ the Graph Attention Network (GAT) [98][2], which uses the self-attention mechanism to specify different weights for different neighboring nodes. However, the vanilla GAT is designed for homogeneous graphs with single-relation edges. To leverage the multi-relational information, we adopt the idea of composition operator [96] to compose entity embeddings and relation embeddings. In detail, in the l-th layer of KM, we update the embedding $E_v^{(l)}$ of entity v as follows: First for each relation entity pair $(r, u) \in \mathcal{N}_v$, we combine the embedding of entity u with the embedding of relation r:

$$M_{u,r}^{(l-1)} = f(E_u^{(l-1)}, R_r) \tag{2.1}$$

Note that the relation embedding R_r is shared across different layers. The function $f(\cdot, \cdot) : \mathbb{R}^F \times \mathbb{R}^F \to \mathbb{R}^F$ merges a pair of entity and relation embeddings into one representation. Here, we set $f(x, y) = x+y$ inspired by TransE [6]. More complicated functions like MLP network can also be applied. Then the combined embeddings are aggregated by graph attention mechanism:

$$m_v^k = \sigma\left(\sum_{(r,u)\in\mathcal{N}_v} \alpha_{v,r,u}^k W^k M_{u,r}^{(l-1)}\right) \tag{2.2}$$

where k is the index of attention head and W^k is the model parameter. The attention score $\alpha_{v,r,u}^k$ is calculated by:

$$S_{u,r} = \mathbf{a}^T \left[W^k E_v^{(l-1)} \oplus W^k M_{u,r}^{(l-1)}\right] \tag{2.3}$$

$$\alpha_{v,r,u}^k = \frac{\exp\left(\text{ LeakyReLU }\left(S_{u,r}\right)\right)}{\sum_{(r',u')\in\mathcal{N}_v} \exp\left(\text{ LeakyReLU }\left(S_{u',r'}\right)\right)} \tag{2.4}$$

Finally the embedding of entity v is updated through combining the message representation m_v^k and its embedding in layer $(l-1)$:

$$E_v^{(l)} = \text{LayerNorm}\left(\bigoplus_{k=1}^{K} m_v^k + E_v^{(l-1)}\right) \tag{2.5}$$

[2]We also tried the vanilla Graph Convolutional Network (GCN) [54] but it performs worse than GAT on the pre-training tasks.

where LayerNorm stands for layer normalization [3]. $\bigoplus$ means concatenation and K is the number of attention heads.

The initial entity embeddings $E^{(0)}$ and relation embeddings R are generated from our language module, which will be introduced in Section "Solving the Cyclic Dependency". Then, the output entity embeddings from the last GAT layer are used as the final entity representations E^{KM}. Note that the knowledge graph can be very large, making the embedding update over all the entities not tractable. Thus we follow the minibatch setting [37]: given a set of input entities, we perform neighborhood sampling to generate their multi-hop neighbor sets and we compute representations only on the entities and relations that are necessary for the embedding update.

2.2.3 Language Module

The goal of the language module (LM) is to model text data and learn context-aware representations. The language module can be any model for language understanding, e.g. BERT [22]. In this work, we use the pre-trained model RoBERTa-base [63] as the language module.

2.2.4 Solving the Cyclic Dependency

In our framework, the knowledge and language modules mutually benefit each other: the language module LM outputs context-aware embedding to initialize the embeddings of entities and relations in the knowledge graph given the description text; the knowledge module (KM) outputs knowledge-based entity embeddings for the language module.

However, there exists a cyclic dependency which prevents computation and optimization in this design. To solve this problem, we propose a decomposed language module which includes two language models: LM_1 and LM_2. We employ the first 6 layers of RoBERTa as LM_1 and the remaining 6 layers as LM_2. The computation proceeds as follows:

1. LM_1 operates on the input text $\mathbf{x}$ and generates contextual embeddings Z.
2. LM_1 generates initial entity and relation embeddings for KM given description text.
3. KM produces its output entity embeddings to be combined with Z and sent into LM_2.
4. LM_2 produces the final embeddings of $\mathbf{x}$, which includes both contextual and knowledge information.

In detail, in step 1, suppose the context $\mathbf{x}$ is embedded as X^{embed}. LM_1 takes X^{embed} as input and outputs hidden representations:

$$Z = \text{LM}_1(X^{embed}) \tag{2.6}$$

In step 2, suppose $\mathbf{x}^{e_j}$ is the *entity description text* for entity e_j, and the corresponding mention is $m^{e_j} = (e_j, s^e_j, o^e_j)$. LM_1 takes the embedding of $\mathbf{x}^{e_j}$ and produces the contextual embedding Z^{e_j}. Then, the average of embeddings at position s^e_j and o^e_j is used as the initial entity embedding of e_j, i.e.

$$E^{(0)}_j = (Z^{e_j}_{s^e_j} + Z^{e_j}_{o^e_j})/2 \tag{2.7}$$

The knowledge graph relation embeddings R are generated in a similar way using its description text.

In step 3, KM computes the final entity embeddings E^{KM}, which is then combined with the output Z from LM_1. In detail, suppose the mentions in $\mathbf{x}$ are $\mathbf{m} = [m_1, \ldots, m_M]$. Z and E^{KM} are combined at positions of mentions: for each position index k, if $\exists i \in \{1, 2, \ldots, M\}$ s.t. $s_{m_i} \leq k \leq o_{m_i}$,

$$Z_k^{merge} = Z_k + E^{\text{KM}}_{e_{m_i}} \tag{2.8}$$

where $E^{\text{KM}}_{e_{m_i}}$ is the output embedding of entity e_{m_i} from KM. For other positions which do not have corresponding mentions, we keep the original embeddings: $Z_k^{merge} = Z_k$. Then we apply layer normalization [3] on Z^{merge}:

$$Z' = \text{LayerNorm}(Z^{merge}) \tag{2.9}$$

Finally, Z' is fed into LM_2.

In step 4, LM_2 operates on the input Z' and obtains the final embeddings:

$$Z^{\text{LM}} = \text{LM}_2(Z') \tag{2.10}$$

The four steps are marked by the symbol ⊗ in Figure 2.2 for better illustration.

2.2.5 Entity Context Embedding Memory

Many knowledge graphs contain a large number of entities. Thus, even for one sentence, the number of entities plus their multi-hop neighbors can grow exponentially with the number of layers in the graph neural network. As a result, it's very time-consuming for the language module to compute context embeddings based on the description text of all involved entities in a batch on the fly.

To solve this problem, we construct an entity context embedding memory, $E^{context}$, to store the initial embeddings of all KG entities. Firstly, the language module pre-computes the context embeddings for all entities and places them into the memory. The knowledge module only needs to retrieve required embeddings from the memory instead of computing them, i.e. $E^{(0)} \leftarrow E^{context}$.

However, as embeddings in the memory are computed from the "old" (initial) language module while the token embeddings during training are computed from the updated language module, there will be an undesired discrepancy. Thus, we propose to update the whole embedding memory $E^{context}$ with the current language module every $T(i)$ steps, where i is the number of times that the memory has been updated (starting from 0). $T(i)$ is set as follows:

$$T(i) = \min(I_{init} * a^{\lfloor i/r \rfloor}, I_{max}) \tag{2.11}$$

where I_{init} is the initial number of steps before the first update and a is the increasing ratio of updating intervals. r is the number of repeated times of the current updating interval. I_{max} is the maximum number of steps between updates. $\lfloor \cdot \rfloor$ means the operation of rounding down. In our

experiments, we set $I_{init} = 10, a = 2, r = 3, I_{max} = 500$, and the corresponding sequence of T is $[10, 10, 10, 20, 20, 20, 40, 40, 40, \ldots, 500, 500]$. Note that we choose $a > 1$ because the model parameters usually change less as training proceeds.

Moreover, we propose a momentum update to make $E^{context}$ evolve more smoothly. Suppose the newly calculated embedding memory by LM is $E^{context}_{new}$, then the updating rule is:

$$E^{context} \leftarrow mE^{context} + (1 - m)E^{context}_{new} \tag{2.12}$$

where $m \in [0, 1)$ is a momentum coefficient which is set as 0.8 in experiment.

This memory design speeds up our model by about 15x during pre-training while keeping the effectiveness of entity context embeddings. For consideration of efficiency, we use relation embeddings only during fine-tuning.

2.2.6 Pre-training

During pre-training, both the knowledge module and language module are optimized based on several self-supervised learning tasks listed below. The examples of all the training tasks are shown in Figure 2.2.

At each pre-training step, we first sample a batch of root entities and perform random-walk sampling on each root entity. The sampled entities are fed into KM for the following two tasks.

Entity category prediction. The knowledge module is trained to predict the category label of entities based on the output entity embeddings E^{KM}. This task has been demonstrated to be effective in pre-training graph neural networks [42]. The loss function is cross-entropy for multi-class classification, denoted as $\mathcal{L}_c$.

Relation prediction. KM is also trained to predict the relation between a given entity pair based on E^{KM}. The loss function is cross-entropy for multi-class classification, denoted as $\mathcal{L}_r$.

Then, we uniformly sample a batch of text sequences and their entities for the following two tasks.

Masked token prediction. Similar to BERT, We randomly mask tokens in the sequence and predict the original tokens based on the output Z^{LM} of the language module. We denote the loss as $\mathcal{L}_t$.

Masked entity prediction. The language module is also trained to predict the corresponding entity of a given mention. For the input text, we randomly remove 15% of the mentions $\mathbf{m}$. Then for each removed mention $m_r = (e_r, s_r, o_r)$, the model predicts the masked entity e_r based on the mention's embedding. In detail, it predicts the entity whose embedding in $E^{context}$ is closest to $q = g((Z^{\text{LM}}_{s_r} + Z^{\text{LM}}_{o_r})/2)$, where $g(x) = \text{GELU}(xW_1)W_2$ is a transformation function. GELU is an activation function proposed by [40]. Since the number of entities can be very large, we use e_r's neighbours and other randomly sampled entities as negative samples. The loss function $\mathcal{L}_e$ is cross entropy based on the inner product between q and each candidate entity's embedding. Figure 2.2 shows an concrete example, where the mention "Earth" is not marked in the input text since it's masked and the task is to link the mention "Earth" to entity "Q2: Earth".

2.2.7 Fine-tuning

During fine-tuning, our model supports using either the knowledge graph employed during pre-training or a novel custom knowledge graph with previously unseen entities[3]. If a custom KG is used, the entity context embedding memory is recomputed by the pre-trained language module using the new entity description text.

Our model also supports KG-only tasks such as entity classification or link prediction where the input data are entity description text and KG without context corpus. In this case, the Language Model 1 takes entity description text as input and output entity embeddings into the knowledge module (i.e. graph neural network) for downstream tasks. The Language Model 2 will not be used.

In this work, we do not update the entity context memory during fine-tuning for consideration of efficiency. We also compute the relation context embedding memory using the pre-trained language model.

2.3 Experiment

2.3.1 Basic Settings

Data for Pre-training. We use the English Wikipedia as the text corpus, Wikidata [103] as the knowledge graph, and SLING [79] to identify entity mentions. For each entity, we use the first 64 consecutive tokens of its Wikipedia page as its description text and we filter out entities without a corresponding Wikipedia page. We also remove entities that have fewer than 5 neighbors in the Wikidata KG and fewer than 5 mentions in the Wikipedia corpus. The final knowledge graph contains 3,657,658 entities, 799 relations and 20,113,978 triplets. We use the *instance of* relation to find the category of each entity. In total, 3,039,909 entities have category labels of 19,901 types. The text corpus contains about 4 billion tokens.

Implementation Details. We initialize the language module with the pre-trained RoBERTa-base [63] model. The knowledge module is initialized randomly. Our implementation is based on the HuggingFace framework [111] and DGL [106]. For the knowledge module, we grid-search the number of layers within [1, 2] (we do not consider more than 2 layers due to model efficiency), the number of attention heads in GAT in [1, 4, 8, 12]. We choose the best performing hyper-parameters based on the validation loss of pre-training tasks after training for 1 epoch: the number of layers is 2 and the number of attention heads is 8. The number of sampled neighbors in each hop is 10. The dimension of hidden states in the knowledge module is 768, the same as the language module. The number of parameters of the whole model is 111M, which is almost the same as RoBERTa-base.

During pre-training, the batch size and length of text sequences are 1024 and 512 respectively. The batch size of KG entities is 16,384. The number of training epochs is 8. JAKET is optimized by AdamW [65] using the following parameters: $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon =$ 1e-8, and weight decay of 0.01. The learning rate of the language module is warmed up over the first 3,000 steps

[3]We assume the custom domain comes with NER and entity linking tools which can annotate entity mentions in text. The training of these systems is beyond the scope of this work.

to a peak value of 1e-5, and then linearly decayed. The learning rate of our knowledge module starts from 1e-4 and then linearly decayed. The computing infrastructure we use is the NVIDIA V100 GPU in all the experiments.

Baselines. We compare our proposed model JAKET with the pre-trained RoBERTa-base [63] and four knowledge-enhanced pre-trained model ERNIE (THU) [129], KnowBERT [71], KEPLER [107] and CoLAKE [90] using their officially released models which are also pre-trained on English Wikipedia corpus and Wikidata KG. We also test two variants of our model: RoBERTa +GNN and RoBERTa+GNN+M. The two models have the same model structure as JAKET, but they are not pre-trained on our data. Moreover, the entity and relation context embedding memories of RoBERTa+GNN are randomly generated while the memories of RoBERTa+GNN+M are computed by RoBERTa.

2.3.2 Downstream Tasks

Few-shot Relation Classification

Relation classification requires the model to predict the relation between two entities in text. Few-shot relation classification takes the N-way K-shot setting. For each query instance, N relations with K supporting examples for each relation are given. The model is required to classify the instance into one of the N relations. In this paper, we evaluate our model on a widely used benchmark dataset FewRel 1.0 [38].

We use the pre-trained knowledge graph for FewRel as it comes with entity mentions from Wikidata knowledge graph. To predict the relation label, we build a sequence classification layer on top of the output of LM. More specifically, we use the PAIR framework proposed by [29], which pairs each query instance with all the supporting instances, concatenate each pair as one sequence, and send the concatenated sequence to our sequence classification model to get the score of the two instances expressing the same relation. We do not use relation embeddings in this task to avoid information leakage.

As shown in Table 2.1, in all three few-shot settings, our model consistently outperforms both ERNIE and KnowBERT, and performs on par with KEPLER. We didn't compare with CoLAKE since its original paper tests on this dataset in a different setting. Comparing the results between RoBERTa and RoBERTa+GNN, we see that adding GNN with randomly generated entity features does not improve the performance. The difference between RoBERTa+GNN+M and RoBERTa+GNN demonstrates the importance of generating context embedding memory by the language module, while JAKET can further improve the performance by pre-training.

Entity Classification

To further evaluate our model's capability to reason over unseen knowledge graphs, we design an entity classification task. Here, the model is given a portion of the Wikidata knowledge graph unseen during pre-training, denoted as $\mathcal{KG}'$. It needs to predict the category labels of these unseen entities. Our entity classification dataset contains a KG with 23,046 entities and 316 relations. The number of triplets is 38,060. Among all the entities, 16,529 of them have category labels and the total number of distinct labels is 1,291. We conduct experiments under

Table 2.1: Accuracy results (mean across 5 different runs) on the dev set of FewRel 1.0. All the models are equipped with the same state-of-the-art few-shot framework PAIR [29].

Model	5-way 1-shot	5-way 5-shot	10-way 1-shot
BERT [22]	85.7	89.5	76.8
ERNIE [129]	86.9	91.4	78.6
KnowBERT [71]	86.2	90.3	77.0
KEPLER [71]	87.3	90.5	**79.4**
RoBERTa [63]	86.4	90.3	77.3
RoBERTa+GNN	86.3	-	-
RoBERTa+GNN+M	86.9	-	-
JAKET	**87.4**	**92.1**	78.9

Table 2.2: Accuracy results (mean across 5 different runs) on the entity classification task over an unseen Wikidata knowledge graph. RoB+G+M is the abbreviation for the baseline model RoBERTa+GNN+M.

Model	Training Size		
	100%	20%	5%
GNN	48.2	-	-
RoBERTa	33.4	-	-
RoB+G+M	79.1	66.7	53.5
JAKET	**81.6**	**70.6**	**58.4**

a semi-supervised transductive setting by splitting the entities in $\mathcal{KG}'$ into train/dev/test splits of 20%, 20% and 60%. To further test the robustness of models to the size of training data, we also evaluate models when using 20% and 5% of the original training set.

In this task, RoBERTa takes the entity description text as input for label prediction while neglecting the structure information of KG. JAKET and RoBERTa+GNN+M make predictions based on the entity representation output from the GNN of the knowledge module. We also include vanilla GNN as a baseline, which uses the same GAT-based structure as our knowledge module, but with randomly initialized model parameters and context embedding memory.

As shown in Table 2.2, our model achieves the best performance under all the settings. The performance of GNN or RoBERTa alone is significantly lower than RoBERTa+GNN+M, which demonstrates the importance of integrating both context and knowledge information using our proposed framework. Also, the gap between JAKET and RoBERTa+GNN+M increases when there's less training data, showing that the joint pre-training can reduce the model's dependence on downstream training data.

2.3.3 Computation Analysis

The computation of the KG module is much less than the LM module. For the RoBERTa-base model, the number of inference computation flops (#flops) over each sequence (length 128) is over 22 billion [92]. Here, we theoretically compute the number of flops of the KG module as follows: The sequence length $N = 128$, and hidden dimension $H = 768$. The number of entities in a sequence is usually less than $N/5$. The number of sampled neighbors per entity $r = 10$. And the number of layers of the GNN based KG module $L = 2$. It follows that the #flops of KG module is about $N/5 \times r^L \times 2H^2 \approx 3$ billion, less than $1/7$ of the language module computation. If we set $r = 5$, the #flops can be further reduced to about $1/30$ of LM computation.

During pre-training, another computation overhead is entity context embedding memory update (Section 3.5): Firstly, the number of entities is about 3 million and the update step interval is about 500. Thus for each step on average the model processes the description text of $3 \times 10^6/500 = 6000$ entities. Secondly, the length of description text is 64, much smaller than the length of input text 512, and we only use LM1 (the first half of LM module) for entity context embedding generation, which saves half of the computation time compared to using the whole LM module. Thirdly, the embedding update only requires forward propagation, costing only half of computation compared to training process which requires both forward and backward propagation. Thus, generating context embedding of 6k entities consumes about the same number of flops as training $6000 \times 64/(512 \times 2 \times 2) \approx 200$ input texts, much smaller than the batch size 1024. In short, the entity context embedding memory update only costs $200/1024 \approx 1/5$ additional computation. Note this computation overhead only exists during pre-training, since entity embedding memory will not be updated during fine-tuning.

2.4 Summary

This chapter presents a novel framework, JAKET, to jointly pre-train models for knowledge graph and language understanding. Under our framework, the knowledge module and language module both provide essential information for each other. After pre-training, JAKET can quickly adapt to unseen knowledge graphs in new domains. Moreover, we design the entity context embedding memory which speeds up the pre-training by 15x. Experiments show that JAKET outperforms baseline methods in KG acquisition tasks including few-shot relation classification and entity classification.

Although effective, JAKET learns vector embeddings for all the entities, leading to high memory usage when dealing with large KGs. This poses a challenge for tasks such as KG completion. In the following chapter, we begin with a light-weight KG completion approach that does not require learning embeddings for all the entities, and propose a novel method to improve it by retrieving relevant triplets from KGs.

Chapter 3

Retrieval-Enhanced Generative Model for KG Completion

3.1 Introduction

Knowledge Graph Completion (KGC), which aims to predict semantically valid but unobserved triplets based on the observed ones, has been a central focus of research in this field. Most studies in KGC have been focusing on relatively small knowledge graphs. For example, the two most common benchmarks namely FB15k-237 [94] and WN18RR [21] contain only 14k and 40k entities, respectively. On the other hand, large KGs in the real world such as Wikidata [102] and Freebase [4] contain millions of entities, which present a scalability challenge to many methods. Traditional KGC methods such as TransE [6] and DistMult [115] have the learn-able embedding parameters for each entity, which means that the number of parameters increases proportionally to the number of entities. Some methods [107, 117] tried to avoid this by employing pre-trained language models (PLM) to embed entities based on their names and/or descriptions. Those methods, however, still suffer from scalability issues at inference due to the need to traverse all the entities for prediction. This means that the inference computation of those methods is linear in the total number of entities.

Recent studies [12, 86] utilize a sequence-to-sequence (seq2seq) generation methodology for knowledge graph completion, by verbalizing each incomplete triplet as the input text sequence and then applying Transformer [97]-based models to predict each missing entity as the output sequence. Those methods are advantageous as the model parameters and inference computation are independent of the size of the knowledge graph. And furthermore, the training of such models does not require negative sampling.

Despite the advantages of existing seq2seq KGC methods, they have one major weakness: That is, their inference only involves implicit reasoning with trained model parameters, neglecting the direct use of the KG itself. In other words, due to the occurrence of catastrophic forgetting during the training process, the models lack the capacity of memorizing all the triplets in large-scale KGs and directly utilizing the triplets for inference. This chapter addresses such limitation by proposing a Retrieval-enhanced Seq2seq KG Completion model, namely ReSKGC. It firstly converts the triplets in the entire KG into text passages, secondly uses a free-text retrieval mod-

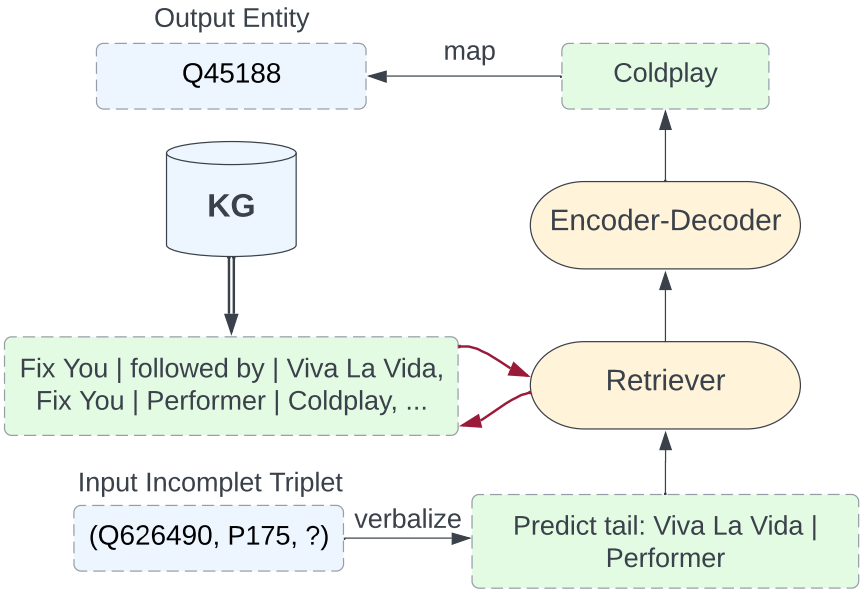


Figure 3.1: Illustration of the proposed framework ReSKGC. Given an incomplete triplet with head entity ID and relation ID, we first verbalize it to a text sequence using their name labels (*Q626490* $\rightarrow$ *Viva La Vida*, *P175* $\rightarrow$ *Performer*), then a retriever is used to retrieve relevant information from the KG, followed by the application of a seq2seq model to generate the name label of the missing entity, which is mapped back into an entity ID (*Coldplay* $\rightarrow$ *Q45188*).

ule (such as BM25 [81]) to find the relevant triplets for each incomplete triplet (the query), and thirdly uses the retrieved triplets to enhance the generation of the missing entity. Figure 4.1 illustrates such a process, with the example of predicting the performer for the song of *Viva La Vida*. Given the query, the retrieved triplets such as *(Fix You, followed by, Viva La Vida)* and *(Fix You, Performer, Coldplay)*, can be used to enhance the likelihood of generating the correct answer.

Our comparative evaluation shows that the proposed approach can significantly enhance the state-of-the-art performance on two large-scale datasets, Wikidata5M [107] and WikiKG90Mv2 [43], which contain 5M and 90M entities, respectively. Ablation tests were also conducted to analyze the effects of different settings in the retrieval module and for relations with different frequencies.

3.2 Preliminary

Knowledge Graph Completion (KGC) In this chapter, KGC refers to the sub-task of missing entity prediction: given an incomplete triplet $(e_h, r, ?)$ or $(?, r, e_t)$ where its head or tail entity is missing, the model is required to predict the missing entity.

Sequence-to-Sequence KGC Given an incomplete triplet, it was first verbalized into a text query

q:

$$q = \begin{cases} \text{concat}(\text{prefix}_t, L(e_h), L(r)) & \text{if } e_t \text{ is missing} \\ \text{concat}(\text{prefix}_h, L(r), L(e_t)) & \text{if } e_h \text{ is missing} \end{cases} \tag{3.1}$$

where $L(\cdot)$ maps the entity or relation into its name label. prefix_t (or prefix_h) refers to a sequence of tokens that are added to the beginning of a query in order to inform the model which side, whether it be the head or tail entity, is the target for prediction. For example, as shown in Figure 4.1, (Q626490, P175, ?) is transformed to *Predict tail: Viva La Vida — Performer*. Similarly, (?, P175, Q45188) will be transformed to *Predict head: Performer — Coldplay*. Then the text query will be passed into an encoder-decoder language model such as T5 [77] to generate the output tokens:

$$T_{\text{output}} = \text{Decoder}(\text{Encoder}(q)) \tag{3.2}$$

The aim is to produce the name label of the missing entity. The cross-entropy loss of token prediction is utilized during the training of the generative model. During the process of making predictions, the generated name label will be mapped to an entity ID $\hat{e}_t = L^{-1}(T_{\text{output}})$. To generate multiple prediction candidates, beam search can be employed, but invalid entity names will be discarded. Constrained decoding methods [12, 20] can be utilized to ensure the validity of the output.

3.3 Method

Instead of directly passing the verbalized incomplete triplet into a seq2seq model for entity generation, we propose ReSKGC, which retrieves relevant information from the knowledge graph to guide the generation process.

3.3.1 KG to Text Passages

To facilitate retrieval based on text semantics of entities and relations, we convert the entire knowledge base into text passages through a process of linearization. To prevent data leakage, we only transform the triplets in the training data. We begin by transforming each triplet into a sentence, accomplished by concatenating the name labels of entities and relations and incorporating a special symbol | to delimit the elements. For example, the triplet *(Q1991309, P175, Q45188)* will be transformed to the sentence *Fix You — Performer — Coldplay.* Subsequently, we group the sentence into passages if they share a common head entity, such as *Fix You — Performer — Coldplay. Fix You — followed by — Viva La Vida...*. Finally, each long passage would be separated into multiple non-overlapping passages, with each passage having a maximum of 100 words.

3.3.2 Retrieval

After converting the knowledge graph into text passages, we proceed to conduct retrieval given an incomplete triplet. To achieve this, we first verbalize the triplet using Equation 3.1. We

then employ the widely-used retrieval method BM25 [81], which is based on TF-IDF scores of sparse word matches between input queries and passages. We have opted for sparse retrieval as opposed to dense retrieval [51], which involves computing passage embeddings and query embeddings using pre-trained language models. The reason for this is that we place value on the generalization and efficiency of sparse representation. Dense retrieval methods typically require additional model training and consume significant amounts of memory in order to save the passage embeddings. Through retrieval, we are able to obtain K passages $[p_i]_{i=1,\cdots,K}$ that are potentially relevant to the input question.

3.3.3 Generation

Ultimately, we employ a generation module that takes the query and retrieved passage as inputs and produces the desired output entity. One approach is to concatenate all the passages and the query as a single input sequence for the model. However, this can become inefficient when a large number of passages are retrieved due to the quadratic computational complexity in the self-attention mechanism of the Transformer model. To achieve both cross-passage modeling and computation efficiency, we apply Fushion-in-Decoder (FiD) [46] based on T5 [77] as the generation module. FiD separately encodes the concatenation of the query and each passage but decodes the output tokens jointly. Specifically, the encoding process for each passage p_i is as follows:

$$\mathbf{P}_i = \text{Encoder}(\text{concat}(q, p_i)). \tag{3.3}$$

Next, the token embeddings of all passages outputted from the encoder are concatenated before being sent to the decoder to produce the output tokens T_{output}:

$$T_{\text{output}} = \text{Decoder}([\mathbf{P}_1; \mathbf{P}_2; \cdots; \mathbf{P}_K]). \tag{3.4}$$

In this manner, the decoder can generate outputs based on joint modeling of multiple passages. Similar to vanilla seq2seq KGC, we utilize beam search to generate multiple candidates and constrained decoding as [12] to ensure validity. After generation, we simply map the name back to the corresponding entity.

3.3.4 Training Process

The training of ReSKGC only involves optimizing the generation module, since the retrieval component (BM25) is unsupervised and does not require any model training. Similar to vanilla seq2seq KGC methods, we employ the cross-entropy loss of token prediction for training. However, there are two crucial considerations:

Removing query triplet from retrieved passages: It is important to note that all of the training triplets are included within the linearized passages as mentioned in Section 3.3.1. Therefore, throughout the training process, it is necessary to ensure that the retrieved passages do not include the query triplet itself, as failure to do so would render the training task insignificant. As a result, we simply remove the linearized triplet from the retrieved passages if it is present in them.

Table 3.1: Dataset Statistics

Dataset	#Entities	#Relations	#Triplets
Wikidata5M	4.8M	828	21M
WikiKG90Mv2	91M	1,387	601M

Sampling query triplets: Large-scale KGs may contain hundreds of millions of triplets, making it excessively costly to enumerate them all during training. In this study, we address this issue by randomly sampling a subset of triplets to generate training queries to optimize the model. We demonstrate in Section 3.4.3 that such sampling does not impair performance while keeping training highly efficient. It should be noted that we still utilize all the training triplets to construct the passages as mentioned in Section 3.3.1.

3.4 Experiment

3.4.1 Basic Setting

We conduct experiments on the two large-scale KGC datasets: Wikidata5M [107], and WikiKG90Mv2 [43], where the dataset statistics are shown in Table 3.1. We follow the conventional train/valid/test split setting according to the original paper of each dataset. For the training process, we use Adam [53] as the optimizer and set the learning rate as 0.0001. The number of training steps is 30,000 for all the datasets with the same batch size of 16. For Wikidata5M, we retrieve 10 passages per query and sampled 100k queries for training. For WikiKG90Mv2, we retrieve 20 passages and sampled 200k queries. During inference, constrained decoding is utilized for the Wikidata5M dataset, while it is not used for WikiKG90Mv2 due to the dataset's extensive number of entities, making the construction of a prefix tree excessively memory and time-consuming. For the generation module, we use T5-small and T5-base [77] with numbers of parameters 60M and 220M, respectively. All experiments are carried out on NVIDIA 2080-Ti GPUs.

For the evaluation metric, we use the filtered setting [6] for computing mean reciprocal rank (MRR) and hits at 1, 3, and 10 (H@1, H@3, and H@10). Higher MRR and hits scores indicate better performance. For Wikidata5M, the metrics are computed for head entity and tail entity prediction separately and then averaged. For WikiKG90Mv2, we follow the original paper to only compute metrics for tail entity prediction.

3.4.2 Main Results

As demonstrated in Table 3.2, ReSKGC attains state-of-the-art results on Wikidata5M, exhibiting a significant enhancement of MRR by 10.6% as compared to the most superior baseline technique. The first section of baselines comprises conventional KGC approaches, which demonstrate satisfactory performance but have a considerably larger number of parameters than the second section, comprising PLM-based techniques. It is noteworthy that SimKGC employs PLM to create embeddings for each entity. Despite having a relatively small number of parameters, it still necessitates substantial memory usage to store all entity embeddings, which is avoided

Table 3.2: KG completion results on Wikidata5M. The best result in each column is marked in bold. The second best is marked in $*$. † results are from the best pre-trained models made available by Graphvite [131]. †† results are from [55]. Other baseline results are from the corresponding papers.

Model	MRR	H@1	H@3	H@10	#Params
TransE [6]†	0.253	0.170	0.311	0.392	2.4B
DistMult [115]†	0.253	0.209	0.278	0.334	2.4B
SimplE [52]†	0.296	0.252	0.317	0.377	2.4B
RotatE [91]†	0.290	0.234	0.322	0.390	2.4B
QuatE [128]†	0.276	0.227	0.301	0.359	2.4B
ComplEx [95]††	0.308	0.255	-	0.398	614M
KEPLER [107]	0.210	0.173	0.224	0.277	125M
MLMLM [17]	0.223	0.201	0.232	0.264	355M
KGT5 [86]	0.300	0.267	0.318	0.365	60M
SimKGC [105]	0.358	0.313	0.376	**0.441**	220M
ReSKGC (small)	0.363*	0.334*	0.386*	0.416	60M
ReSKGC (base)	**0.396**	**0.373**	**0.413**	0.437*	220M

by our proposed methodology. It is also noteworthy that our method outperforms the baselines significantly on Hits@1, with an improvement of 19.1%, but it performs slightly worse than the best baseline SimKGC on Hits@10. We posit that the generation-based approach utilized by our model may produce less diverse predicted answers in contrast to the matching-based approach. Thus, having more predictions through the matching approach can result in better improvement of answer coverage.

Table 3.3 presents results on the extremely large-scale dataset WikiKG90Mv2[1], which contains 90 million entities. Our proposed method surpasses the current state-of-the-art technique[2] by 13.2%, while utilizing only 1% of the parameters. Moreover, our method outperforms the seq2seq KGC baseline KGT5, even when both models have an equivalent number of parameters.

3.4.3 Ablation Study

In this section, we aim to answer the following essential questions for a more comprehensive understanding of our proposed method.

Q1. How does the number of retrieved passages affect the model performance? First, we demonstrate the efficacy of retrieval through the variation in the number of passages retrieved, ranging from 0 to 20, where 0 implies the application of vanilla seq2seq KGC without retrieval. Figure 3.2 indicates that retrieval yields considerable advantages on both datasets. Notably, the retrieval of 10 passages can increase the MRR by 16.1% and 89.2% for the respective datasets when compared to non-retrieval. Furthermore, we note that enhancing the number of passages

[1]We only show results obtained from the validation set since the test set is not publicly available.

[2]We conduct a fair comparison by solely considering a single model, without any ensemble methods.

Table 3.3: KG completion results on WikiKG90Mv2 (validation set). The best result is marked in bold. The second best is marked in $*$. All the baseline results are taken from the official leaderboard of [43] except that † results are from [86].

Model	MRR	#Params
TransE-Shallow-PIE [11]	0.234*	18.2B
TransE-Concat [43]	0.206	18.2B
ComplEx-Concat [43]	0.205	18.2B
ComplEx-MPNet [43]	0.126	307K
ComplEx [95]	0.115	18.2B
TransE-MPNet [43]	0.113	307K
TransE [6]	0.110	18.2B
KGT5 [86]†	0.221	60M
ReSKGC (small)	0.230	60M
ReSKGC (base)	**0.265**	220M

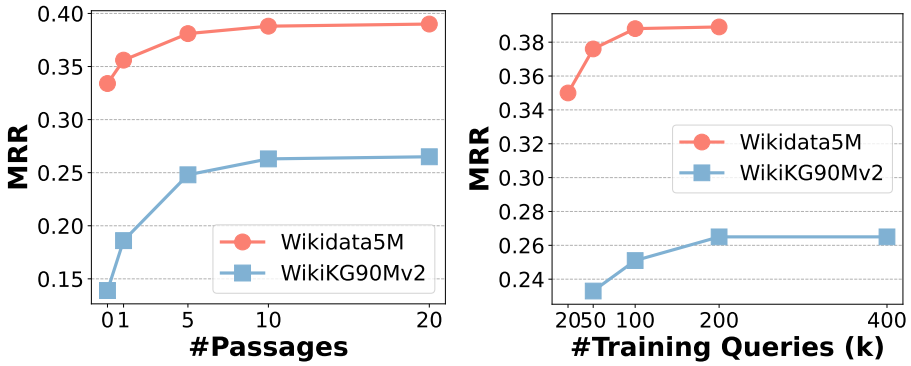


Figure 3.2: The performance of ReSKGC (base) over the validation sets based on different numbers of retrieval passages and sampled training queries.

from 10 to 20 produces minimal performance enhancement, indicating that the information presented in later passages is overshadowed by noise. Therefore, an additional increase in the number of retrieved passages to the generation module will not lead to a significant improvement in performance.

Q2. How does the sampling of training queries affect the final performance? As mentioned in Section 3.3.4, we randomly sample a subset of triplets as training queries. Figure 3.2 illustrates the performance of the model with varying numbers of triplets used as training queries. Our results indicate that for Wikidata5M, a mere 100K triplets are sufficient to achieve good performance, which represents less than 0.5% of the total number of triplets. For WikiKG90Mv2, 200K triplets are sufficient, representing only 0.03% of the total triplets. These findings suggest that a retrieval-enhanced model can effectively learn patterns from a considerably smaller amount of training data.

Q3. What's the effect of retrieval on relations with different frequencies? In a knowledge

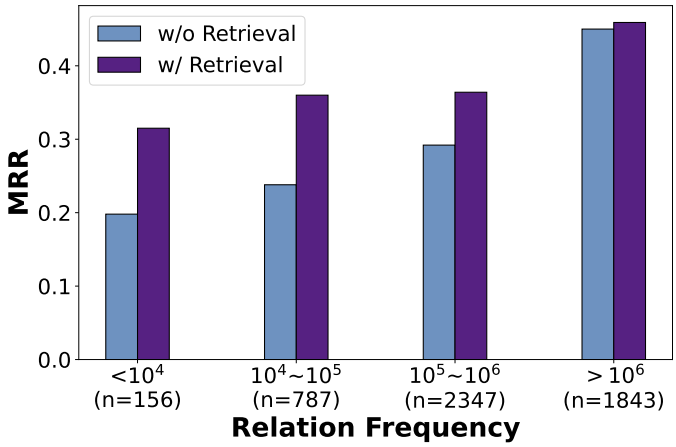


Figure 3.3: The performance of ReSKGC (base) on the Wikidata5M test set triplets, categorized according to their relation frequencies. n represents the number of test triplets within that particular range of relation frequencies.

graph, relations manifest in variable quantities of triplets. To classify relations by their frequency, we have grouped them into four categories based on the number of occurrences: less than 10^4, 10^4 to 10^5, 10^5 to 10^6, and more than 10^6. Figure 3.3 illustrates that the incorporation of retrieval yields more pronounced gains for relations that occur less frequently. An illustrative example can be observed in the relations that occur less than 10^4 times. In this scenario, the inclusion of retrieval results in a 59% MRR increment. In contrast, for relations that appear in more than 10^6 triplets, the MRR increment is merely 2% when utilizing retrieval. This finding is logical, as the generation module may lack adequate training data to comprehend relations that are less commonly observed. Therefore, retrieval-enhanced explicit reasoning can facilitate performance improvement in such cases.

3.5 Summary

This chapter presents a Retrieval-enhanced Seq2seq KG Completion model, namely ReSKGC, which converts the triplets in the entire KG into text passages and uses a free-text retrieval module to find the relevant triplets for each incomplete triplet. The retrieved triplets are then used to enhance the generation of the missing entity. Comparative evaluations on two large-scale datasets, Wikidata5M and WikiKG90Mv2, demonstrate that the proposed approach significantly outperforms the state-of-the-art models by 10.6% and 13.2% in terms of mean reciprocal rank. Additionally, we conducted ablation studies to explore the effects of the number of retrieved passages, training data, and relation frequencies. Our approach shows a promising direction for large-scale KG completion, where further improvements can be achieved by research on improving the retrieval modules and generation modules.

Now we conclude our exploration of KG acquisition using textual semantics, by proposing a joint pre-training framework of both KG and text, and a retrieval-enhanced KG completion method by transforming triples to text. In the next part, we'll explore how to leverage textual

semantics for the application of KGs to question answering. We'll start with the scenarios where answers are sourced solely from the KGs, then expand to cases where answers are primarily drawn from textual data. Lastly, we'll bridge KG acquisition and application by proposing a new benchmark studying the effects of KG completion over question answering.

Part II

Answering Text Question with KGs

Chapter 4

Joint Generation of Answers and Logical Queries

4.1 Introduction

Knowledge Graph Question Answering (KGQA) aims to answer natural language questions based on knowledge from KGs such as DBpedia [2], Freebase [5] or Wikidata [102]. Existing methods can be divided into two categories. One category is based on semantic parsing, where models first parse the input question into a logical form (e.g., SPARQL [41] or S-expression [33]) then execute the logical form against KGs to obtain the final answers [18, 33, 118]. The other category of methods directly outputs answers without relying on the the logical-form executor [70, 86, 88]. They either classify the entities in KGs to decide which are the answers [88] or generate the answers using a sequence-to-sequence framework [70, 86].

Previous empirical results [18, 35, 118] show that the semantic parsing based methods can produce more accurate answers over benchmark datasets. However, due to the syntax and semantic restrictions, the output logical forms can often be non-executable and thus would not produce any answers. On the other hand, direct-answer-prediction methods can guarantee to generate output answers, albeit their answer accuracy is usually not as good as semantic parsing based methods, especially over complex questions which require multi-hop reasoning [93]. To our knowledge, none of the previous studies have leveraged the advantages of both types of methods. Moreover, since KGs are usually large-scale with millions of entities, most previous methods rely on entity linking to select relevant information from KGs for answering questions. However, these entity linking methods are usually designed for specific datasets, which inevitably limits the generalization ability of these methods.

In this chapter, we propose a novel framework DECAF to overcome these limitations:

(1) Instead of relying on only either logical forms or direct answers, DECAF jointly decodes them together, and further combines the answers executed using logical forms and directly generated ones to obtain the final answers. Thus the advantages of both methods can be leveraged in our model. Moreover, unlike previous methods using constrained decoding [15] or post revision [18] to produce more faithful logical forms, we simply treat logical forms as regular text strings just like answers during generation, reducing efforts of hand-crafted engineering.

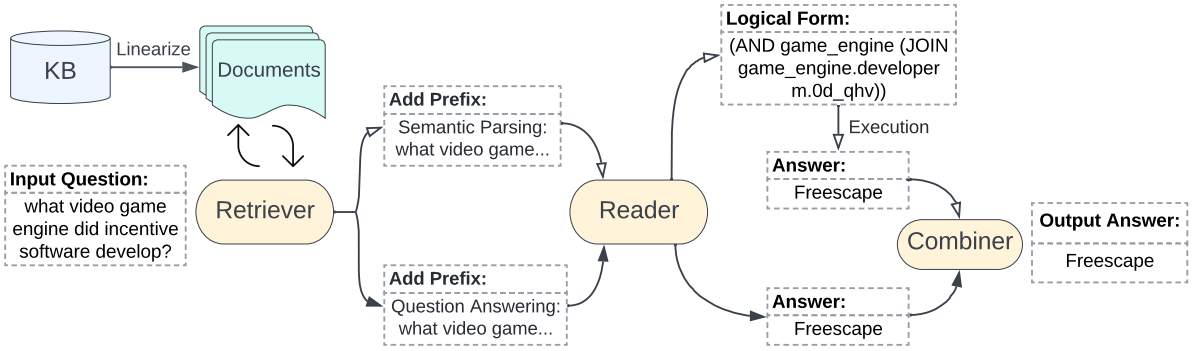


Figure 4.1: Model framework of DECAF. We use text-based retrieval instead of entity linking to select question-related information from the KG. Then, we add different prefixes into the reader to generate logical forms and direct answers respectively. The logical-form-executed answers and directly-generated answers are combined to obtain the final output.

(2) Different from previous methods which rely on entity linking [60, 119] to locate entities appeared in questions and then retrieve relevant information from KG, DECAF linearizes KGs into text documents and leverages free-text retrieval methods to locate relevant sub-graphs. Based on this simplification, DECAF brings better adaptation to different datasets and potentially different KGs due to the universal characteristic of text-based retrieval. Experiments show that simple BM25 retrieval brings surprisingly good performance across multiple datasets.

We conduct experiments on four benchmark datasets including WebQSP [120], ComplexWebQuestions [93], FreebaseQA [47], and GrailQA [33]. Experiment results show that our model achieves new state-of-the-art results on WebQSP, FreebaseQA, and GrailQA benchmarks, and gets very competitive results on the ComplexWebQuestions benchmark. This demonstrates the effectiveness of DECAF across different datasets and question categories.

4.2 Method

In order to (1) leverage the advantages of both logical forms and direct answers and (2) reduce the dependency on entity linking models, we propose a novel framework DECAF. As shown in Figure 1, the whole KG is first transformed into text documents. Then, for an input question, the retriever retrieves relevant passages from linearized KG documents, which will be combined with the input question into the reader. DECAF reader is a sequence-to-sequence generative model and uses different prefixes to generate logical forms (LFs) and direct answers respectively. Finally, the executed answers by LFs and generated direct answers are combined to obtain the final answers.

4.2.1 KG Linearization

Given a question, retrieving relevant information from KGs is non-trivial since KGs are large-scale and complicated with both semantic (names of entities and relations) and structural (edge

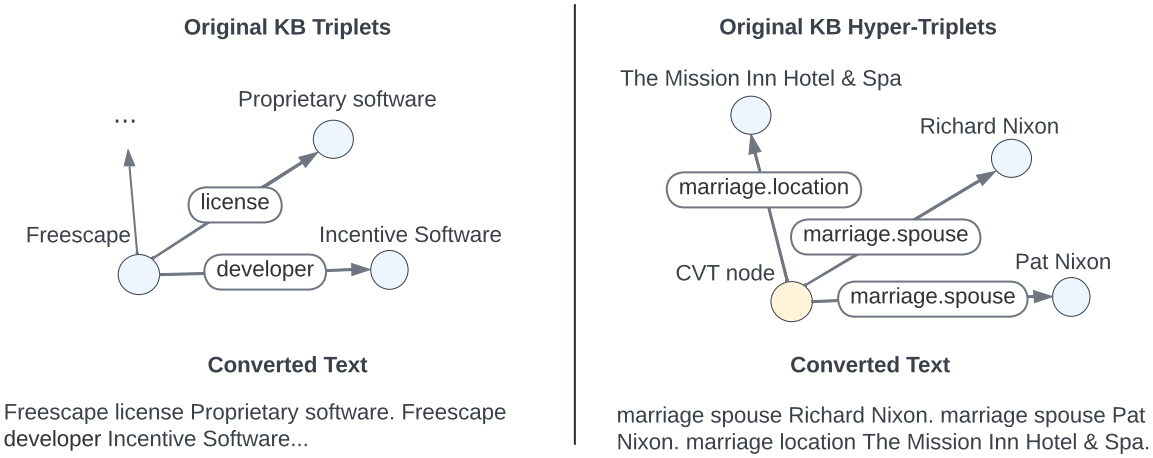


Figure 4.2: KG linearization. We show examples of how we linearize triplets (two entities and one relation) and hyper-triplets (multiple entities and relations with a central CVT node).

between entities by relations) information. On the other hand, recent studies have shown the effectiveness of text-based retrieval for question answering [13, 46, 51]. By linearizing KGs to text corpus, we can easily leverage these successful text-retrieval methods to select semantically relevant information from KGs.

We describe how to linearize the knowledge graph. Considering the original KG format to be the most common Resource Description Format (RDF), which contains triplets composed of one head entity, relation, and tail entity. For example, (*Freescape*, *game_engine.developer*, *Incentive Software*) means that Incentive Software is the developer of a game engine called Freescape. To linearize this triplet, we simply concatenate them with spaces to be a sentence as *Freescape game engine developer Incentive Software.* Note that we also replace punctuation marks in relation to spaces. In this case, this sentence contains both semantic information (entity and relation names) and structural information (relation between two entities). Then we further group sentences with the same head entity to become a document like *Freescape game engine developer Incentive Software. Freescape release date 1987....* This grouping is mainly to preserve the structural information of the 1-hop subgraph corresponding to the head entity. Following [51], we truncate each document into multiple non-overlap passages with a maximum of 100 words.

Note that triplets have semantic constrains to express complicated relations. For example, it is hard to express that Richard Nixon and Pat Nixon got married in The Mission Inn Hotel & Spa, which involves three entities. In Freebase, this is solved by introducing a new node called CVT node, which serves as a connecting entity for such hyper-triplets but has no meaning (name) itself. In this example, an entity with id *m.02h98gq* is introduced which involves triplets (*m.02h98gq*, *marriage.spouse*, *Pat Nixon*), (*m.02h98gq*, *marriage.spouse*, *Richard Nixon*), and (*m.02h98gq*, *marriage.location_of_ceremony*, *The Mission Inn Hotel & Spa*). In this case, instead of concatenating CVT node id into sentence, we ignore this node while grouping other entities and relations into one passage: *marriage spouse Richard Nixon. marriage spouse Pat Nixon. marriage location of ceremony The Mission Inn Hotel & Spa.* We illustrate our KG linearization

in Figure 4.2. In the next section, we show how to conduct retrieval from these passages.

4.2.2 Retrieval

The retriever retrieves relevant passages from the linearized KG based on the input question. We consider two kinds of retrieval methods: sparse retrieval and dense retrieval. For sparse retrieval, we use BM25 [81], which is based on TF-IDF scores of sparse word match between input questions and KG-linearized passages. For dense retrieval, we apply the DPR [51] framework, which is based on similarity in the embedding space between input questions and passages from two fine-tuned BERTs [23]. We refer readers to the original paper for details of the fine-tuning process. During inference, suppose there are totally N passages in the knowledge source $\{p_1, p_2, \ldots, p_N\}$. DPR applies the passage encoder $E_P(\cdot)$ to encode all the passages and store embeddings in memory. For an input question q, DPR applies the question encoder $E_Q(\cdot)$ to obtain its representation, and then the passages are retrieved based on the dot-product similarity: $I_{\text{retrieve}} = \text{argtop-k}_i(E_P(p_i) \cdot E_Q(q))$. Then it applies FAISS[49] to conduct an efficient similarity search due to the large number of passages N. Through this step, we can retrieve $|I_{\text{retrieve}}| \ll N$ passages which are potentially relevant to the input question.

4.2.3 Reading

The reader takes the retrieved passages and the original question as input and generates the targeted output. Recent advanced sequence-to-sequence models like BART [58] and T5 [77] can be utilized here. In order to answer complicated multi-hop questions, cross-passage reasoning is important for the reader. One way is to concatenate all the passages and let the self-attention in the Transformer module capture these patterns. However, this can be inefficient when the number of retrieved passages is very large because of the quadratic computation complexity in self-attention. To achieve both cross-passage modeling and computation efficiency, we apply Fushion-in-Decoder (FiD) [46] based on T5 as the reader model. FiD encodes the concatenation of the input question and each passage separately but decodes the output tokens jointly. Specifically, we denote the question as q and the retrieved passages as $\{p_{r_i} | r_i \in I_{\text{retrieve}}\}$. The encoding process for each passage p_{r_i} is:

$$\mathbf{P}_i = \text{Encoder}(\text{concat}(q, p_{r_i})) \in \mathbb{R}^{L_p \times H}, \tag{4.1}$$

where H is the hidden dimension, and L_p is the total sequence length of a question concatenated with a passage. T5-Encoder$(\cdot)$ is the encoder module of T5. Then the token embeddings of all passages output from the last layer of the encoder are concatenated and sent to the decoder to generate the output tokens T_{output}:

$$T_{\text{output}} = \text{Decoder}([\mathbf{P}_1; \mathbf{P}_2; \cdots; \mathbf{P}_{|I_{\text{retrieve}}|}] \in \mathbb{R}^{|I_{\text{retrieve}}| L_p \times H}) \tag{4.2}$$

By concatenating the encoder output embeddings, the decoder can generate outputs based on joint modeling of multiple passages.

4.2.4 Joint Decoding Answers and Logical Forms

Motivated by the success of adding prefixes to control the generation of large-scale language models [77, 112], we use a shared sequence-to-sequence model to generate both logical forms and direct answers, and differentiate the two processes by prompting the model with different prefixes. The encoding-decoding process in Equation (5.5) and (5.6) becomes:

$$\mathbf{P}_i^{\text{answer}} = \text{Encoder}(\text{concat}(\text{prefix}^{\text{answer}}, q, p_{r_i})),\ T_{\text{answer}} = \text{Decoder}([\mathbf{P}_1^{\text{answer}}; \cdots; \mathbf{P}_{|I_{\text{retrieve}}|}^{\text{answer}}]); \quad (4.3)$$

$$\mathbf{P}_i^{\text{LF}} = \text{Encoder}(\text{concat}(\text{prefix}^{\text{LF}}, q, p_{r_i})),\ T_{\text{LF}} = \text{Decoder}([\mathbf{P}_1^{\text{LF}}; \cdots; \mathbf{P}_{|I_{\text{retrieve}}|}^{\text{LF}}]) \quad (4.4)$$

where T_{answer} and T_{LF} are the output answer tokens and logical form tokens respectively. $\text{prefix}^{\text{answer}}$ and $\text{prefix}^{\text{LF}}$ are the prefixes for answer generation and logical form generation respectively, which we set as *Question Answering:* and *Semantic Parsing:*. For example, given the question *What video game engine did incentive software develop?*, we'll first retrieve relevant passages using the retriever. Then we add these two prefixes to produce two different inputs: *Question Answering: what video game engine did incentive software develop?* and *Semantic Parsing: what video game engine did incentive software develop?"* For the first input, we train the model to generate the target output answer *Freescape* directly[1]. While for the second input, we aim to generate the logical form `(AND cvg.computer_game_engine (JOIN cvg.computer_game_engine.developer m.0d_qhv))`, which is simply treated as text strings during generation without constrained decoding. However, instead of directly generating the entity ID like *m.0d_qhv* in the logical form, we replace it with the corresponding entity name like *Incentive Software* and add special tokens "[" and "]" to identify it. The revised logical form to be generated becomes `(AND cvg.computer_game_engine (JOIN cvg.computer_game_engine.developer [ Incentive Software ]))`. After generation, we replace the entity names with their IDs for execution. During training, we fine-tune the whole reader in a multi-task learning manner, where one input question contributes to two training data pairs with one for answer generation and the other one for logical form generation. During inference, we use beam search to generate multiple candidates for both logical forms and answers, with the same beam size B.

Next, we show how we combine them to obtain the final answer. We first execute these logical forms against the KG using an executor[2]. Suppose the list of executed answer set is $[A_1^{\text{LF}}, \cdots, A_{B'}^{\text{LF}}]$ ($B' \leq B$ since some logical forms can be non-executable) and the list of directly generated answer set is $[A_1^{\text{answer}}, \cdots, A_B^{\text{answer}}]$. We consider two situations: (1) If $B' = 0$ means none of these logical forms are executable, the final answer is simply A_1^{answer}; (2) Otherwise when $B' \geq 1$, we use weighted linear combination: we first assign the score $\lambda S(k)$ for A_k^{LF} and the score $(1-\lambda)S(k)$ for A_k^{answer}, where $0 \leq \lambda \leq 1$ is a hyper-parameter controlling the weight of each type of answers, and $S(k)$ is a score function based on the answer rank k. If an answer set appears both in executed answer list and generated answer list with ranks i and j respectively, then its score is the sum of two scores: $\lambda S(i) + (1-\lambda)S(j)$. Finally, we select the answer set with the highest score as the final output. We leave the exploration of other combination methods as future work.

[1] For direct answer generation, we only consider returning one single answer.

[2] We locally set up a Virtuoso triplestore service following the GrailQA paper.

Model	WebQSP		CWQ	FreebaseQA
	Hits@1	F1	Hits@1	Hits@1
PullNet [88]	67.8	62.8	47.2	-
EmQL [89]	75.5	-	-	-
NSM$_{+h}$ [39]	74.3	-	53.9	-
FILM [101]	54.7	-	-	63.3
KGT5 [86]	56.1	-	36.5	-
CBR-SUBG [19]	72.1	-	-	52.1
SR+NSM [126]	69.5	64.1	50.2	-
UniK-QA [70]	79.1	-	-	-
QGG [57]	-	74.0	44.1	-
HGNet [16]	70.6	70.3	65.3	-
ReTrack [15]	74.7	74.6	-	-
CBR-KBQA [18]	-	72.8	**70.4**	-
ArcaneQA [32]	-	75.3	-	-
Program Transfer [10]	74.6	76.5	58.1	-
RnG-KBQA [118]	-	75.6	-	-
RnG-KBQA (T5-large)*	-	76.2 ± 0.2	-	-
DECAF (BM25 + FiD-large)	79.0 ± 0.4	74.9 ± 0.3	68.1 ± 0.5	78.8 ± 0.5
DECAF (DPR + FiD-large)	80.7 ± 0.2	77.1 ± 0.2	67.0 ± 0.4	**79.0** ± 0.6
DECAF (BM25 + FiD-3B)	-	-	**70.4**	-
DECAF (DPR + FiD-3B)	**82.1**	**78.8**	-	-

Table 4.1: Results on the test splits of 3 benchmark datasets: WebQSP, CWQ, and FreebaseQA. The two blocks of baselines are direct-answer-prediction and semantic parsing based methods respectively. We run 5 independent experiments for FiD-large based DECAF and report mean and standard deviation. * means that we replace the original reader T5-base with T5-large and rerun experiments to have a fair comparison with our method.

4.3 Experiment

Experiment Settings. We use the full Freebase [5] data pre-processed by [109] as the KG for all benchmarks. The total number of entities, relations, and triplets are about 88 million, 20k, and 472 million respectively. The total number of passages after linearization is about 126 million. For the retrieval module of DECAF, we use BM25 implemented by Pyserini [61] and Dense Passage Retrieval (DPR) [51] with BERT-base [23] as question and passage encoders. We train separate DPRs on each dataset following the same training process and use the same model architecture in the original paper. The number of retrieved passages is 100 if not specified. For the reading module, we leverage Fusion-in-Decoder [46] based on the T5 [77] model: FiD-large and FiD-3B with 770 million and 3 billion model parameters respectively. For the decoding beam size, we use 10 for FiD-large model and 15 for FiD-3B model.

We evaluate DECAF on four benchmark datasets: WebQSP [120], ComplexWebQuestions (CWQ) [93], FreebaseQA [47], and GrailQA [33]. Specifically, GrailQA provides the categories of questions: i.i.d., compositional, and zero-shot, which can be used to evaluate model perfor-

Model	Overall	I.I.D.	Compositional	Zero-Shot
QGG [57]	36.7	40.5	33.0	36.6
Bert+Ranking [33]	58.0	67.0	53.9	55.7
ReTrack [15]	65.3	87.5	70.9	52.5
S2QL (Anonymous)	66.2	72.9	64.7	63.6
ArcaneQA [32]	73.7	-	75.3	66.0
RnG-KBQA [118]	74.4 (76.9)	89.0 (88.3)	71.2 (69.2)	69.2 (75.1)
RnG-KBQA (T5-large)*	- (77.1)	- (88.5)	- (69.8)	- (75.2)
UniParser (Anonymous)	74.6	-	71.1	69.8
DeCC (Anonymous)	77.6	-	75.8	72.5
TIARA (Anonymous)	78.5	-	76.5	**73.9**
DECAF (BM25 + FiD-large)	76.0 (78.7)	**90.5** (90.2)	79.0 (78.7)	68.0 (73.7)
DECAF (DPR + FiD-large)	- (75.4)	- (89.7)	- (75.8)	- (69.0)
DECAF (BM25 + FiD-3B)	**78.7** (81.4)	89.9 (89.7)	**81.8** (80.1)	72.3 (78.4)

Table 4.2: F1 scores on the test split of GrailQA. The numbers in the parentheses are F1 scores on the dev split. * means that we replace the original reader T5-base with T5-large and rerun experiments.

mance over different levels of generalization. All datasets provide both ground-truth answers and logical forms except FreebaseQA, where our model only generates answers as the final output without using the logical form. More details about these datasets are shown in the appendix. Following previous work [18, 70, 118], we evaluate our model based on metrics Hits@1 and F1, where Hits@1 focus on the single top-ranked answer while F1 also considers coverage of all the answers.

4.3.1 Main Result

We compare with both direct answer prediction and semantic parsing based methods and run 5 independent experiments for FiD-large based DECAF to report mean and standard deviation. We don't do this for DECAF with FiD-3B due to the limitation of computation resource. We denote our model in the form of DECAF ({Retriever} + {Reader}) such as DECAF (BM25 + FiD-large). If the retriever is not specified, it means we choose the better one for each dataset respectively.

As shown in Table 4.1, DECAF achieves new SOTA on WebQSP and FreebaseQA dataset, outperforming both direct-answer-prediction (first block) and semantic parsing based (second block) baselines. On WebQSP, DECAF improves the previous highest Hits@1 by 3.0% and F1 by 2.3%. Note that one of the best-performing baseline UniK-QA uses STAGG [119] for entity linking on WebQSP, which is not publicly available and thus can not be applied to other datasets. Compared to UniK-QA, we see that DECAF (DPR + FiD-large) improves the Hits@1 by 1.6% with the same model size, demonstrating the effectiveness of our method even without entity linking. On FreebaseQA, DECAF improves the SOTA Hits@1 significantly by 15.7%. On CWQ dataset, DECAF achieves very competitive results, the same as the current SOTA method CBR-KBQA. CBR-KBQA is based on the K-Nearest Neighbors approach, which is

GrailQA / λ	0.0	0.2	0.4	0.45	0.49	0.51	0.55	0.6	0.8	1.0
$S(k) = 1/k$	54.7	54.7	56.1	56.3	56.5	78.7	78.7	78.7	78.7	78.7
$S(k) = B - k + 1$	54.7	55.6	56.4	56.5	56.5	78.3	78.3	78.4	78.6	78.7
WebQSP / λ	0.0	0.2	0.4	0.45	0.49	0.51	0.55	0.6	0.8	1.0
$S(k) = 1/k$	49.8	49.8	50.6	51.0	51.3	76.7	76.7	76.9	77.1	77.1
$S(k) = B - k + 1$	49.8	51.0	51.5	51.8	51.9	75.2	75.2	75.2	76.0	77.1
CWQ / λ	0.0	0.2	0.4	0.45	0.49	0.51	0.55	0.6	0.8	1.0
$S(k) = 1/k$	50.5	50.5	52.7	53.5	54.4	68.5	68.5	68.6	68.6	68.6
$S(k) = B - k + 1$	50.5	53.1	54.3	54.6	54.6	68.3	68.3	68.4	68.6	68.6

Table 4.3: F1 scores on GrailQA and WebQSP and Hits@1 scores on CWQ using DECAF (FiD-large) based on different values of λ, which is the weight of LF-executed answers in the answer combination function. $S(k)$ is the score function of answer rank k and B is the generation beam size.

complementary to our method. We also see that increasing reader size from large (770M) to 3B significantly improves model performance. Moreover, BM25 and DPR lead to different results: DPR performs significantly better than BM25 on WebQSP, slightly better on FreebaseQA, and worse on CWQ. The possible reason is that DPR is trained to retrieve passages containing the answers based on distant supervision [51], which do not necessarily contain the relations and entities that appeared in the logical form, especially for complex questions. Thus it may hurt the performance of logical form generation which results in a final performance degeneration.

Table 4.2 shows results on the GrailQA dataset, where we listed F1 scores of Overall, I.I.D., Compositional, and Zero-shot questions respectively. We see that with FiD-large reader, DECAF achieves better overall performance than the published SOTA method RnG-KBQA. Since original RnG-KBQA uses T5-base, we also equip it with T5-large for a fair comparison and list the results on the dev set, where DECAF (BM25 + FiD-large) still significantly outperforms it by 1.6% in overall F1 score. With FiD-3B reader, the overall F1 of DECAF is improved by 2.7% compared to using FiD-large reader, and surpasses the current SOTA method TIARA[3], which is an anonymous submission on the GrailQA leaderboard. Notably, DECAF performs the best in compositional questions with an F1 score 5.3% higher than the best-performing method.

Overall we see that DECAF achieves new SOTA results on WebQSP, FreebaseQA, and GrailQA with very competitive results on the CWQ benchmark. Even with simple BM25 retrieval, DECAF outperforms most of the baselines across all the benchmark datasets, which previous studies usually use different entity linking methods specially designed for different datasets.

4.3.2 Ablation Study

In this section, we conduct some ablation studies to answer the following questions:
What is the best way to combine LF-executed answers and generated answers? In Section 4.2.4, we introduced a way of combining LF-executed answers and generating answers to obtain

[3]We achieve the 1st rank on the GrailQA leaderboard as of 09/06/2022.

Model	WebQSP		CWQ	GrailQA (dev)			
	Hit@1	F1	Hit@1	F1 (O)	F1 (I)	F1 (C)	F1 (Z)
DECAF (Answer only)	74.7	49.8	50.5	54.7	59.4	38.3	59.5
DECAF (LF only)	74.3	74.0	55.2	72.4	88.2	76.3	63.9
DECAF	80.7	77.1	68.1	78.7	90.2	78.7	73.7
Non-Executable LF%	11.3		30.2	15.8	6.2	9.6	22.5
DECAF$_{sep}$ (Answer only)	74.2	49.5	47.9	54.6	57.7	38.8	59.8
DECAF$_{sep}$ (LF only)	72.7	73.1	54.3	74.0	91.4	75.2	66.0
DECAF$_{sep}$	80.6	77.1	66.5	80.3	92.9	78.9	75.3
Non-Executable LF%	12.3		32.8	15.9	5.2	11.9	22.4

Table 4.4: We study the performance of our model when only using generated answers (Answer Only) or executed answers by logical forms (LF Only). O, I, C, Z means overall, i.i.d., compositional and zero-shot. DECAF$_{sep}$ means using a separate reader for answer generation and logical form generation respectively instead of a joint reader. We also show the percentage of questions where none of the generated LFs are executable.

the final answer. (1) When none of the LFs are executable, we use the top-1 generated answer as output. In Table 4.4, we show the percentage of questions where none of the generated LFs are executable. It can be observed that directly generating LFs for hard questions, such as CWQ and Zero-shot GrailQA, shows a significantly higher non-executable rate than that for easy questions. (2) If any LF is executable, we use the answer with the highest combination scores $\lambda S(i) + (1 - \lambda)S(j)$, where λ is the hyper-parameter weight for LF-executed answers, and i and j are the rank in the LF-executed answer list and generated answer list respectively. We test two different score functions $S(k) = 1/k$ and $S(k) = B - k + 1$ where B is the beam size, and k is the rank in the candidate list. From the results in Table 4.3, we see that the model performance is improved when λ increases. Specifically (1) the F1 score increases dramatically when λ changes from 0.49 to 0.51, where the former usually chooses generated answer as a final answer while the latter selects the LF-executed one. This demonstrates that LF-executed answers are much more accurate compared to generated ones, which can also be validated in the next section. (2) $\lambda = 1.0$ gives the best results, which means that we can simply select the first executed answer if exists, otherwise choose the first direct answer. Thus we set $\lambda = 1.0$ as the default setting of DECAF in all the experiments.

How do answer generation and LF generation perform respectively? DECAF combines generated answers and executed answers to obtain the final answers. In this section, we study the impact of them separately. We introduce two new models to (1) only use generated answers (Answer Only) and (2) only use executable answers by logical form (LF Only). As shown in the first group of models in Table 4.4, the performance of LF Only is consistently better than Answer Only. Except on WebQSP, the latter has similar Hits@1, but the former has a significantly better F1 score, which shows that LF can produce much better answer coverage. We also see that the combination of these two strategies is significantly better than any of them separately. For example, on the dev set of GrailQA, the overall F1 of DECAF is 6.3% higher than DECAF (LF only) and 24.0% higher than DECAF (Answer Only).

Should we use a joint reader or separate readers for answer and LF generation? We add

Model	WebQSP		CWQ		GrailQA		FreebaseQA	
	H@100	R@100	H@100	R@100	H@100	R@100	H@100	R@100
BM25	81.2	67.7	63.5	57.7	89.9	84.7	93.5	93.5
DPR	91.6	80.6	71.4	65.6	87.6	81.0	95.0	95.0

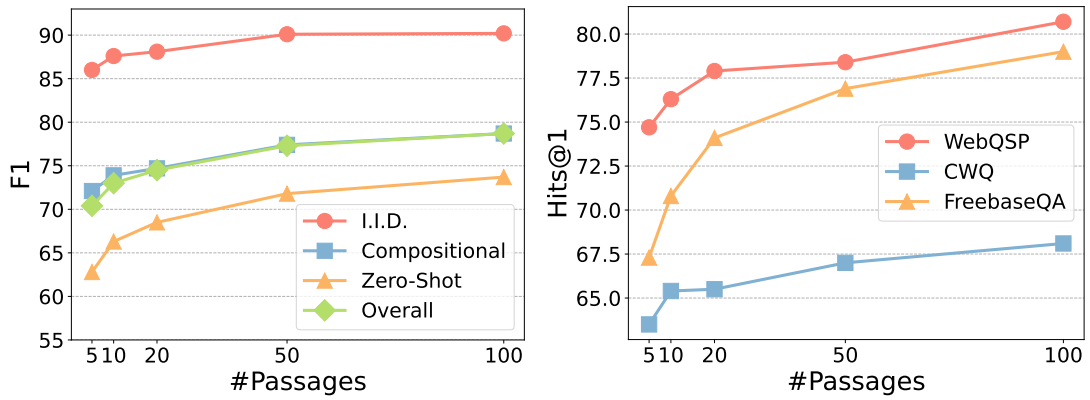
Table 4.5: Retrieval results (answer name match). H@100 and R@100 stand for the answer hits rate and recall rate of 100 passages, respectively.

another variation of our model called DECAF$_{sep}$, which uses two separate readers to generate direct answers and logical forms respectively instead of a shared reader, while other parts remain the same as DECAF. As shown in the second group of models in Table 4.4, we see that the overall performance is similar to DECAF, showing that a shared reader with prefix is as capable as two separate readers. However, it is interesting to see that on CWQ, a shared reader with multi-task training performs better on answer generation and LF generation while on GrailQA, it performs worse on LF generation compared to two separate readers.

Does retrieval capture useful information for answering questions? We first evaluate the retrieval results of BM25 and DPR on 4 datasets in Table 4.5. We observe that: (1) DPR performs better than BM25 on all the datasets except GrailQA, which contains zero-shot questions where DPR may have poor performance due to generalization issues. (2) On WebQSP, GrailQA and FreebaseQA, the retrieval method can achieve over 80% hits rate and recall rate, demonstrating the effectiveness of retrieval. (3) The performance is not as good on CWQ dataset, where most questions require multi-hop reasoning. This problem can be potentially mitigated by iterative retrieval [72, 113], which we leave as future work. We then analyze the effect of retrieval on the final QA performance in Figure 4.3, where we show the results over 4 datasets with different numbers of retrieved passages. We see that on all the datasets, with the increase of passage number (from 5 to 100), the model performance is improved. Specifically, on GrailQA dataset, over different categories of questions, we see that the performance over I.I.D. questions increases the least while it improves the most over zero-shot questions. This is because I.I.D. questions can be handled well by memorizing training data while zero-shot questions require external knowledge from KG to be well answered.

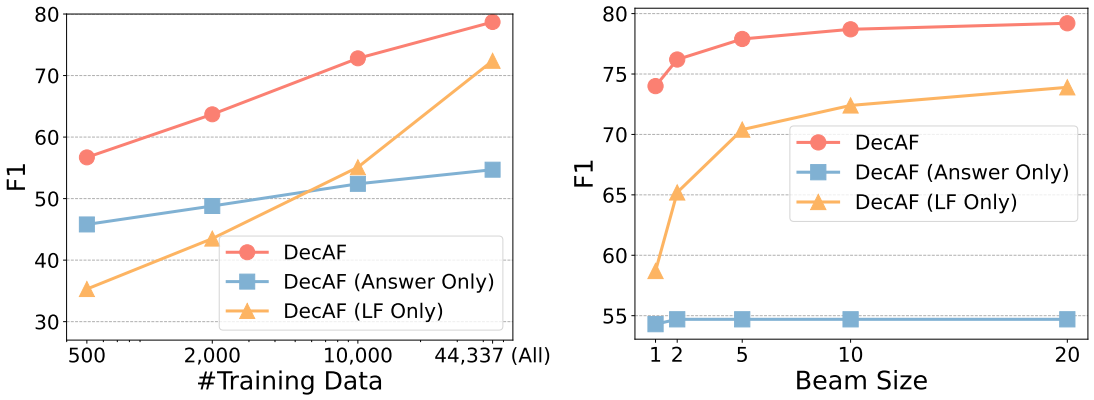
How does the size of training data affect the model performance? In this section, we further study the influence of the size of the training data. Specifically, we want to study how it influences answer generation and LF generation respectively. We focus on the GrailQA dataset, and vary the number of training data from 500, 2000, 10000 to 44337 (all). As shown in Figure 4.4(a), the performance of DECAF improves as the increase of training data. More importantly, we see that the performance of LF generation improves much more significantly than DECAF (Answer Only). This shows that training the logical form generation requires more data than answer generation, because logical forms are usually more complicated than answers.

During inference, what is the effect of beam size? In this section, we study the effects of generation beam size during inference. As shown in Figure 4.4(b), when we vary the beam size from 1,2,5,10 to 20, the performance of DECAF improves. However, we see that the performance DECAF (LF Only) is significantly improved while DECAF (Answer Only) barely changes. This shows that the overall performance improvement mainly comes from logical form generation instead of answer generation. This is because the logical form is difficult to generate, and beam



(a) Results on GrailQA. (b) Results on WebQSP, CWQ, and FreebaseQA

Figure 4.3: DECAF (FiD-large) performance based on different number of retrieval passages.



(a) Results based on different sizes of training data (b) Results based on different beam sizes

Figure 4.4: Ablation study on training data size and generation beam size over GrailQA (dev) dataset.

size is important to long sequence generation especially when we enumerate the generated logical forms until we find the one that is executable. However, for answer generation, which is usually in short length, beam size won't have a large effect.

4.3.3 Error Analysis

We conduct case-based error analysis on our model. As shown in Table 4.6, we list 3 cases where Answer Only generation is wrong, LF Only generation is wrong, and both of them are wrong. In the first example, we need to find the "earlies" composition. We see that the logical form is not complicated while the direct answer generation needs to reason over the completion time of compositions and find the minimum of them, which is difficult. In the second example, we see that the reasoning is not difficult since it only contains one relation about locomotive class, but the logical form is relatively long which makes the generation more challenging. We see that the generated logical form misses the *steam* part and results in an non-executable prediction. For

Question:	*Which composition was completed the earliest?*
Gold Answer:	*Ce fut en mai*
Gold Logical Form:	`(ARGMIN music.composition music.composition.date_completed)`
Gen Answer:	*Composition for Piano and Orchestra by David Bowie*
Gen Logical Form:	`(ARGMIN music.composition music.composition.date_completed)`
LF Executed Answer:	*Ce fut en mai*
DECAF's Answer:	*Ce fut en mai*
Question:	*British rail class 04 belongs to which locomotive class?*
Gold Answer:	*0-6-0*
Gold Logical Form:	`(AND rail.steam_locomotive_wheel_configuration (JOIN rail.steam_locomotive_wheel_configuration. locomotive_classes m.02rh__)))`
Gen Answer:	*0-6-0*
Gen Logical Form:	`(AND rail.locomotive_wheel_configuration (JOIN rail.locomotive_wheel_configuration. locomotive_classes m.02rh__)))`
LF Executed Answer:	*Not Executable*
DECAF's Answer:	*0-6-0*
Question:	*Which browser was most recently released by the creators of mpd?*
Gold Answer:	*Internet Explorer for Mac*
Gold Logical Form:	`(ARGMAX (AND computer.web_browser (JOIN (R computer. software_developer.software) (JOIN (R computer. file_format.format_creator) m.0210900))) computer.software.first_released)`
Gen Answer:	*WebKit*
Gen Logical Form:	`(ARGMAX (AND computer.web_browser (JOIN (R computer. software_developer.software) m.0210900)) computer.software.release_date)`
LF Executed Answer:	*Not Executable*
DECAF's Answer:	*WebKit*

Table 4.6: Case-based error analysis over GrailQA (dev) dataset, where Gen is the abbreviation for Generated. We show 3 cases where only generated answer is wrong, only generated LF is wrong, and both of them are wrong.

these two cases, DECAF can still output the correct answer due to answer combination. However, this is not the case for the last example, which is a compositional question involving multiple relations. We see that both the generated answer and logic form are wrong. The generated logical form neglects one join operation and makes mistake on the relation about release date. This means that our model can be further improved to deal with such complicated questions.

4.4 Summary

In this chapter, we present a novel method DECAF to jointly generate direct answers and logical forms (LF) for knowledge graph question answering. We found that combining the generated answers and LF-executed answers can produce more accurate final answers. Instead of relying on entity linking, DECAF is based on a sequence-to-sequence framework enhanced by retrieval, where we transform the knowledge graph into text and use sparse or dense retrieval to select relevant information from KG to guide output generation. Experimental results show that we achieve the new state-of-the-art on WebQSP, FreebaseQA, and GrailQA. Our work sheds light on the relationship between more general semantic parsing based methods and direct-answer-prediction methods. It would be interesting to further explore this direction on other tasks involving both semantic parsing and end-to-end methods like table-based question answering or programming code generation.

The joint generation of direct answers and logical forms in DecAF is both robust and effective, but it's limited to cases where answers come from KGs. What happens when answers are sourced from different mediums, like text corpora? In the following chapter, we'll demonstrate that in such settings, we can still utilize both KGs and textual semantics to enhance question-answering performance.

Chapter 5

KG-Enhanced Passage Reranking for Answer Generation

5.1 Introduction

Open-Domain Question Answering (ODQA) is the task of answering natural language questions from open-domain text corpora. A successful ODQA model relies on effective acquisition of world knowledge. A popular line of work treats a large collection of open-domain documents (such as Wikipedia articles) as the knowledge source, and design a ODQA system that consists of a *retrieving module* and a *reading module*. The retriever pulls out a small set of potentially relevant passages from the open-source documents for a given question, and the reader produces an answer based on the retrieved passages [36, 45, 51]. An earlier example of this kind is DrQA [14], which used an traditional search engine based on the bag of words (BoW) document representation with TF-IDF term weighting, and a neural reader for extracting candidate answers for each query based on the dense embedding of the retrieved passages. With the successful development of Pre-trained Language Models (PLMs) in neural network research, dense embedding based passage retrieval (DPR) models [51, 74] have shown superior performance over BoW/TF-IDF based retrieval models due to utilization of contextualized word embedding in DPR, and generative QA readers [59, 80] usually outperform extraction based readers [24, 36] due to the capability of the former in capturing lexical variants with a richer flexibility.

The recently proposed Fusion-in-Decoder (FiD) model [46] is representative of those methods with a DPR retriever and a generative reader, achieving the state-of-the-art results on ODQA evaluation benchmarks. FiD also significantly improved the scalability of the system over previous generative methods by encoding the retrieved passages independently instead of encoding the concatenation of all retrieved passages (which was typical in previous methods).

Inspired by the success of FiD, this chapter aims further improvements of the state of the art of ODQA in the paradigm with a DPR retriever and a generative reader. Specifically, we point out two potential weaknesses or limitations of FiD as the rooms for improvements, and we propose a novel solution namely KG-FiD to address these issues with FiD. The two issues are:

Issue 1. The independent assumption among passages is not justified. Notice that both the DPR retriever and the generative reader of FiD perform independent encoding of the retrieved

passages, which means that they cannot leverage the semantic relationship among passages for passage embedding and answer generation even if such relational knowledge is available. But we know that rich semantic connections between passages often provide clues for better answering questions [67].

Issue 2. Efficiency Bottleneck. For each input question, the FiD generative reader receives about 100 passages from the DPR module, with a relatively high computational cost. For example, the inference per question takes more than 6 trillion floating-point operations. Simply reducing the number of retrieved passages sent to the reader will not be a good solution as it will significantly decrease the model performance [46]. How to overcome such inefficient computation issue is a challenging question for the success of FiD in realistic ODQA settings.

We propose to address both of the above issues with FiD by leveraging an existing knowledge graph (KG) to establish relational dependencies among retrieved passages, and employing Graph Neural Networks (GNNs) to re-rank and prune retrieved passages for each query. We name our new approach as KG-FiD.

Specifically, KG-FiD employs a two-stage passage reranking by applying GNN to model structural and semantic information of passages. Both stages rerank the input passages and only a few top-reranked passages are fed into subsequent modules. The first stage reranks passages returned by the retriever, where we use the passage embeddings generated by DPR as the initial GNN node representation. This allows reranking a much larger set of initial candidate passages to enhance coverage of answers. The second stage performs joint passage reranking and answer generation, where the node embeddings are initialized by the embeddings of passage-question pairs output from the reader encoder. This stage operates on a smaller candidate set but aims for more accurate reranking and passage pruning.

To improve the efficiency, in the second-stage reranking, our GNN model adopts representations from the intermediate layer in the reader encoder instead of the final layer to initiate passage node embeddings. Then only a few top reranked passages will be passed into the higher layers of encoder and the decoder for answer generation, while other passages will not be further processed. This is coupled with a joint training of passage reranking and answer generation. As shown in Section 5.3.4, these strategies significantly reduce the computation cost while still maintaining a good QA performance.

Our experiments on ODQA benchmark datasets Natural Questions and TriviaQA demonstrate that KG-FiD can achieve comparable or better performance in answer prediction than FiD, with only 40% of the computation cost of FiD.

5.2 Method

In the following sections, we first introduce how to apply KG to build a graph structure among the retrieved passages (Section 5.2.1). Then we show how we adopt the graph-based stage-1 reranking with DPR retriever to improve passage retrieval (Section 5.2.2). Next we introduce joint stage-2 reranking and answer generation in the reading module (Section 5.2.3). Finally we illustrate the improvement of efficiency by using intermediate layer representation for stage-2 reranking (Section 5.2.4). The overview of our framework is illustrated in Figure 5.1.

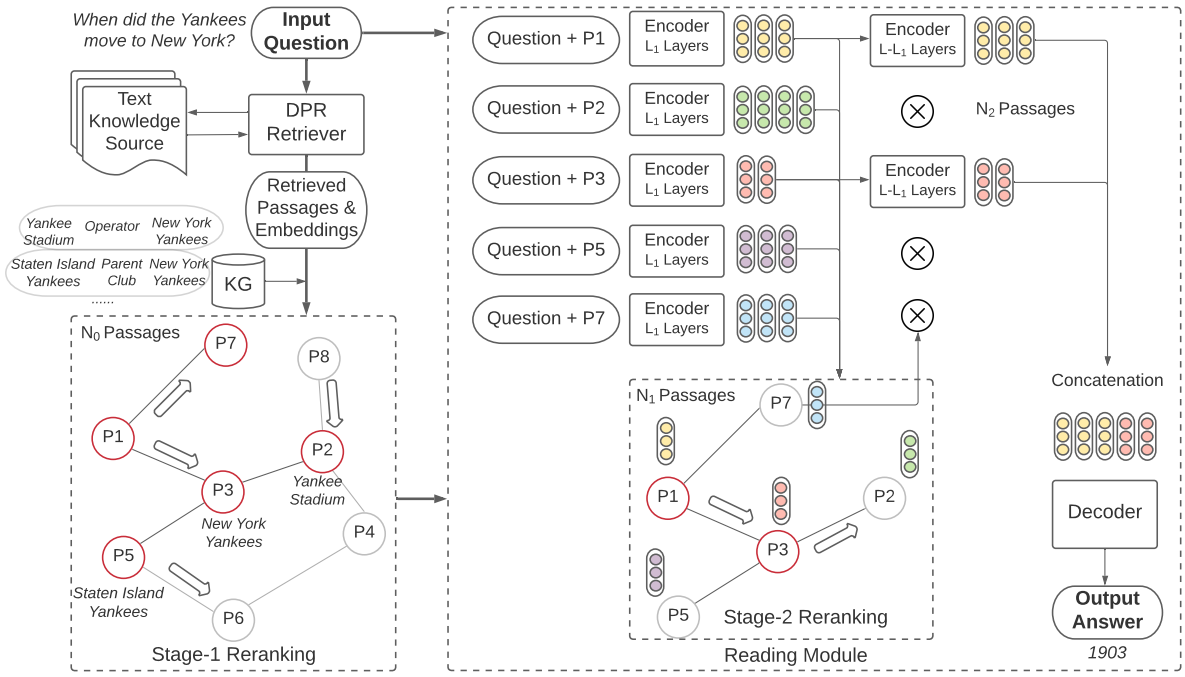


Figure 5.1: Overall Model Framework. Pi indicates the node of the passage originally ranked the i-th by the DPR retriever, with the article title below it. The left part shows passage retrieval by DPR, passage graph construction based on KG (Section 5.2.1) and stage-1 reranking (Section 5.2.2). The right part shows joint stage-2 reranking and answer generation in the reading module (Section 5.2.3 and 5.2.4).

5.2.1 Construct Passage Graph using KG

The intuition behind using KG is that there exists the structural relationship among the retrieved passages which can be captured by the KG. Similar to [67], we construct the passage graph where vertices are passages of text and the edges represent the relationships that are derived from the external KGs as $\mathcal{KG} = \{(e_h, r, e_t)\}$, where e_h, r, e_t are the head entity, relation and tail entity of a triplet respectively.

First, we formalize the definition of a *passage*. Following previous works [51, 108], each article in the text corpus is split into multiple disjoint text blocks of 100 words called *passages*, which serve as the basic retrieval units. We assume there is a one-one mapping between the KG entities and articles in the text corpus. Specifically, we use English Wikipedia as the text corpus and English Wikidata [102] as the knowledge graph, since there exists an alignment between the two resources[1]. For example, for the article titled with "New York Yankees", it contains passages such as "The New York Yankees are an American professional baseball team ...". The article also corresponds to a KG entity with the same name as "New York Yankees".

Then we define the mapping function $e = f(p)$, where the KG entity e corresponds to the article which p belongs to. Note that one passage can only be mapped to one entity, but multiple passages could be mapped to the same entity. The final passage graph is defined as $\mathcal{G} = \{(p_i, p_j)\}$, where passages p_i and p_j are connected if and only if their mapped entities are directly connected in the KG, i.e., $(f(p_i), r, f(p_j)) \in \mathcal{KG}$.

Since the total number of passages is very large, e.g., more than 20M in Wikipedia, constructing and maintaining a graph over all the passages is inefficient and memory-consuming. Thus, we build a passage graph on the fly for each question, based on the retrieved passages.

5.2.2 Passage Retrieving & Stage-1 Reranking

DPR Retriever: Our framework applies DPR [51] as the retriever, which uses a BERT based passage encoder to encode all the N passages in the text corpus $\{p_1, p_2, \cdots, p_N\}$. Suppose all the passage embeddings are fixed and stored in memory as $M \in \mathbb{R}^{N \times D}$ where D is the hidden dimension:

$$M_i = \text{BERT}(p_i) \text{ for } i \in \{1, 2, \cdots N\} \tag{5.1}$$

For an input question q, DPR applies another BERT-based question encoder to obtain its representation Q, then it builds on FAISS [49] to conduct fast dot-product similarity search between Q and M, and returns N_1 ($N_1 \ll N$) passages with the highest similarity scores.

Stage-1 Reranking: We see that the DPR retriever returns N_1 passages which are independently retrieved based on the similarity between the question and each passage, without considering inter-passage relationship. Thus instead of directly retrieving N_1 passages for the reader, we propose to first retrieve N_0 ($N_0 > N_1$) passages, then rerank them and output top-N_1 reranked passages into the reader.

Following Section 5.2.1, we construct a graph among the N_0 retrieved passages denoted as $\mathcal{G}_0$. We aim to rerank the retrieved passages based on both the structural information and the textual semantic information of them.

[1]Entity recognition and linking can be used if there is no such alignment.

To represent the semantic information of passages, one can use another pre-trained language model to encode the passage texts, but this will not only include lots of additional model parameters, but also incur heavy computational cost as N_0 can be large. To avoid both additional memory and computation cost, we propose to reuse the offline passage embeddings M generated from the DPR retriever in Equation 5.1 as the initial node representation: $E_i^{(0)} = M_{r_i}$ where $\{r_i | i \in \{1, 2, \cdots, N_0\}\}$ is the set of retrieved passage indices.

Then we employ a graph attention network (GAT) [98] with L_g layers as GNN model to update representations for each node based on the passage graph and initial representation. The l-th layer of the GNN model updates the embedding of node i as follows:

$$E_i^{(l)} = h(E_i^{(l-1)}, \{E_j^{(l-1)}\}_{(i,j)\in\mathcal{G}_0}) \tag{5.2}$$

where h is usually a non-linear learnable function which aggregates the embeddings of the node itself and its neighbor nodes. The reranking score for each passage p_{r_i} is calculated by $s_i^{\text{stage-1}} = Q^T E_i^{(L_g)}$, where Q is the question embedding also generated by the DPR retriever. Then we sort the retrieved passages by the reranking scores, and input the top-N_1 passages into the reader. The training loss of passage ranking for each question is:

$$\mathcal{L}_r^{\text{stage-1}} = -\sum_{i=1}^{N_0} y_i \log\left(\frac{\exp(s_i^{\text{stage-1}})}{\sum_{j=1}^{N_0}\exp(s_j^{\text{stage-1}})}\right) \tag{5.3}$$

where $y_i = 1$ if p_{r_i} is the gold passage[2] that contains the answer, and 0 otherwise.

As we only add a lightweight graph neural network and reuse the pre-computed and static DPR passage embeddings, our reranking module can process a large number of candidate passages efficiently for each question. In experiments, we set $N_0 = 1000$ and $N_1 = 100$.

5.2.3 Joint Stage-2 Reranking and Answer Generation

In this section, we briefly introduce the vanilla FiD reading module before illustrating our joint reranking method. We suppose the reader takes N_1 retrieved passages $\{p_{a_1}, p_{a_2}, \cdots, p_{a_{N_1}}\}$ as input.

Vanilla FiD Reading Module: We denote the hidden dimension as H and number of encoder layers and decoder layers as L, FiD reader first separately encodes each passage p_{a_i} concatenated with question q:

$$\mathbf{P}_i^{(0)} = \text{T5-Embed}(q + p_{a_i}) \in \mathbb{R}^{T_p \times H}, \tag{5.4}$$

$$\mathbf{P}_i^{(l)} = \text{T5-Encoder}_l(\mathbf{P}_i^{(l-1)}) \in \mathbb{R}^{T_p \times H}, \tag{5.5}$$

where T_p is the sequence length of a passage concatenated with the question. T5-Embed$(\cdot)$ is the initial embedding layer of T5 model [77] and T5-Encoder$_l(\cdot)$ is the l-th layer of its encoder module. Then the token embeddings of all passages output from the last layer of the encoder are concatenated and sent to the decoder to generate the answer tokens $\mathbf{A}$:

$$\mathbf{A} = \text{T5-Decoder}\left([\mathbf{P}_1^{(L)}; \mathbf{P}_2^{(L)}; \cdots; \mathbf{P}_{N_1}^{(L)}]\right) \tag{5.6}$$

[2] We follow Karpukhin et al. [51] on the definition of gold passages.

Stage-2 Reranking: Note that vanilla FiD reader neglect the cross information among passages, and the joint modeling in the decoding process makes it vulnerable to the noisy irrelevant passages. Thus, we propose to leverage the passage graph to rerank the input N_1 passages during the encoding and only select top-N_2 ($N_2 < N_1$) reranked passages into the decoder, which is named as stage-2 reranking.

Similar to stage-1 reranking, the reranking model is based on both the structural information and the textual semantic information of passages. We denote the passage graph as $\mathcal{G}_1$, which is a subgraph of $\mathcal{G}_0$. To avoid additional computation and memory cost, we propose to reuse the encoder-generated question-aware passage representation from FiD reader for passage reranking as it is already computed in Equation 5.5. Specifically, the initial node embeddings $Z_i^{(0)}$ for passage p_{a_i} comes from the first token embedding of the final layer in the FiD-Encoder, i.e., $Z_i^{(0)} = \mathbf{P}_i^{(L)}(0) \in \mathbb{R}^D$. Then same as stage-1 reranking, we also employ a GAT [98] with L_g layers as the graph neural network (GNN) model to update representations for each node based on the passage graph, similar to Equation 5.2: $Z^{(L_g)} = \text{GAT}(Z^{(0)}, \mathcal{G}_1')$. The reranking score of passage p_{a_i} is calculated by $s_i^{\text{stage-2}} = W^T Z_i^{(L_g)}$ where W is a trainable model parameter. After reranking, only the final top-N_2 ($N_2 < N_1$) passages are sent for decoding. Suppose their indices are $\{g_1, g_2, \cdots, g_{N_2}\}$, the decoding process in Equation 5.6 becomes:

$$\mathbf{A} = \text{T5-Decoder}\left([\mathbf{P}_{g_1}^{(L)}; \mathbf{P}_{g_2}^{(L)}; \cdots; \mathbf{P}_{g_{N_2}}^{(L)}]\right) \tag{5.7}$$

where $\mathbf{A}$ is the generated answer. Similar to stage-1 reranking, the training loss of passage ranking for each question is:

$$\mathcal{L}_r^{\text{stage-2}} = -\sum_{i=1}^{N_1} y_i \log\left(\frac{\exp(s_i^{\text{stage-2}})}{\sum_{j=1}^{N_1}\exp(s_j^{\text{stage-2}})}\right) \tag{5.8}$$

where $y_i = 1$ if p_{a_i} is the gold passage that contains the answer, and 0 otherwise.

The passage reranking and answer generation are jointly trained. We denote the answer generation loss for each question is $\mathcal{L}_a$, then the final training loss of our reader module is $\mathcal{L} = \mathcal{L}_a + \lambda \mathcal{L}_r^{\text{stage-2}}$, where λ is a hyper-parameter which controls the weight of reranking task in the total loss.

Note that the first stage reranking is based on DPR embeddings, which are are high-level (one vector per passage) and not further trained. While the second stage is based on reader-generated passage-question embeddings, which are semantic-level and trainable as part of the model output. Thus the second stage can better capture semantic information of passages and aims for more accurate reranking over a smaller candidate set. In the experiment, we set $N_1 = 100$ and $N_2 = 20$.

5.2.4 Improving Efficiency via Intermediate Representation in Stage-2 Reranking

Recall that in the stage-2 reranking, we take the passage representation from the last layer of reader encoder for passage reranking. In this section, we propose to further reduce the computation cost by taking the intermediate layer representation rather than the last layer. The intuition is

that answer generation task is more difficult than passage reranking which only needs to predict whether the passage contains the answer or not. Thus we may not need the representation from the whole encoder module for passage reranking.

Suppose we take the representation from the L_1-th layer ($1 \leq L_1 < L$), i.e., $Z_i^{(0)} = \mathbf{P}_i^{(L_1)}(0)$ for $i \in \{1, 2, \cdots, N_1\}$, and the reranking method remains the same. Then only the top N_2 ($N_2 < N_1$) reranked passages will go through the rest layers of FiD-encoder. Suppose their indices are $I_g = \{g_1, g_2, \cdots, g_{N_2}\}$, for $l \geq L_1 + 1$:

$$\mathbf{P}_i^{(l)} = \begin{cases} \text{T5-Encoder}_l(\mathbf{P}_i^{(l-1)}) & \text{if } i \in I_g \\ \text{Stop-Computing} & \text{else} \end{cases} \tag{5.9}$$

Then $\mathbf{P}_{g_1}^{(L)}, \mathbf{P}_{g_2}^{(L)}, \cdots, \mathbf{P}_{g_{N_2}}^{(L)}$ are sent into the decoder for answer generation as in Equation 5.7. In Section 5.3.4, we demonstrate this can reduce 60% computation cost than the original FiD while keeping the on-par performance on two benchmark datasets.

5.2.5 Analysis on Computational Complexity

Here we analyze the theoretical time complexity of our proposed KG-FiD compared to vanilla FiD. More practical computation cost comparison is shown in Appendix 5.3.6. Because both the computations of DPR retrieving and stage-1 reranking are negligible compared to the reading part, we only analyze the reading module here.

Suppose the length of answer sequence $\mathbf{A}$ is denoted as T_a and the average length of the passage (concatenated with question) is T_p. For vanilla FiD reader, the time complexity of the encoder module is $O(L \cdot N_1 \cdot T_p^2)$, where L, N_1 denote the number of encoder layers and the number of passages for reading. The square comes from the self-attention mechanism. The decoder time complexity is $O(L \cdot (N_1 \cdot T_p \cdot T_a + T_a^2))$, where $N_1 \cdot T_p \cdot T_a$ comes from the cross-attention mechanism. For our reading module, all the N_1 candidate passages are processed by the first L_1 layers of encoder. But only N_2 passages are processed by the remaining $L - L_1$ encoder layers and sent into the decoder. Thus, the encoder computation complexity becomes $O((L_1 \cdot N_1 + (L - L_1) \cdot N_2) \cdot T_p^2)$, and the decoder computation takes $O(L \cdot (N_2 \cdot T_p \cdot T_a + T_a^2))$. Because $L_1 < L, N_2 < N_1$, both the encoding and decoding of our method is more efficient than vanilla FiD.

Furthermore, the answer is usually much shorter than the passage (which is the case in our experiments), i.e., $T_a \ll T_p$. Then the decoding computation can be negligible compared to the encoding. In this case, the approximated ratio of saved computation cost brought by our proposed method is:

$$\begin{aligned} S &= 1 - \frac{(L_1 \cdot N_1 + (L - L_1) \cdot N_2) \cdot T_p^2}{L \cdot N_1 \cdot T_p^2} \\ &= (1 - \frac{L_1}{L})(1 - \frac{N_2}{N_1}) \end{aligned}$$

This shows that we can reduce more computation cost by decreasing L_1 or N_2. For example, if setting $L_1 = L/4, N_2 = N_1/5$, we can reduce about 60% of computation cost. More empirical results and discussions will be presented in Section 5.3.4.

5.3 Experiment

In this section, we conduct extensive experiments on two most commonly-used ODQA benchmark datasets: Natural Questions (NQ) [56] which is based on Google Search Queries, and TriviaQA [50] which contains questions from trivia and quiz-league websites. The open-domain version of NQ is obtained by discarding answers with more than 5 tokens. For TriviaQA, its *unfiltered* version is used for ODQA. We follow the same setting as [46] to preprocess these datasets. We convert all letters of answers in lowercase except the first letter of each word on TriviaQA. When training on NQ, we sample the answer target among the given list of answers, while for TriviaQA, we use the unique human-generated answer as generation target. For both datasets, we use the original validation data as test data, and keep 10% of the training set for validation. All our experiments are conducted on 8 Tesla A100 40GB GPUs.

5.3.1 Implementation Details

Knowledge Source: Following [46, 51], we use the English Wikipedia as the text corpus, and apply the same preprocessing to divide them into disjoint passages with 100 words, which produces 21M passages in total. For the knowledge graph, we use English Wikidata. The number of aligned entities, relations and triplets among these entities are 2.7M, 974 and 14M respectively.

Model Details: For the retrieving module, we use the DPR retriever [51] which contains two BERT (base) models for encoding question and passage separately. For the GNN reranking models, we adopt 3-layer Graph Attention Networks (GAT) [98]. For the reading module, same as [46], we initialize it with the pretrained T5-base and T5-large models [77], and we name the former one as KG-FiD (base) and the latter one as KG-FiD (large). Our implementation is based on the HuggingFace Transformers library [110]. For number of passages, we set $N_0 = 1000, N_1 = 100, N_2 = 20$.

Model Training: For training our framework, we adopt the separate-training strategy to avoid out-of-memory issue: we first train the DPR model following its original paper, then freeze the DPR model to train the stage-1 reranking module, and finally jointly train stage-2 reranking and reader part. For the training of stage-1 reranking, the optimizer is AdamW [65] with learning rate as 1e-3 and linear-decay scheduler. The weight decay rate is 0.01. Batch size is set as 64. The number of total training steps is 15k, and the model is evaluated every 500 steps and the model with best validation results is saved as the final model. For the training of reading part, we adopt the same training setting except that the learning rate is 1e-4 for the base model and 5e-5 for the large model. We also adopt learning rate warm up with 1000 steps.

Evaluation: We follow the standard evaluation metric of answer prediction in ODQA, which is the exact match score (EM) [78]. A generated answer is considered correct if it matches any answer in the list of acceptable answers after normalization[3]. For all the experiments, we conduct 5 runs with different random seeds and report the averaged scores.

[3]The normalization includes lowercasing and removing articles, punctuation and duplicated whitespace.

5.3.2 Baseline Methods

We mainly compare KG-FiD with the baseline model FiD [46]. For other baselines, we compare with representative methods from each category: (1) not using external knowledge source: T5 [80] and GPT-3 [8]; (2) reranking-based methods: RIDER [66] and RECONSIDER [44]; (3) leveraging knowledge graphs or graph information between passages: Graph-Retriever [67], Path-Retriever [1], KAQA [130], and UniK-QA [68]. We also compare with methods (4) with additional large-scale pre-training: REALM [36], RAG [59] and Joint Top-K [83].

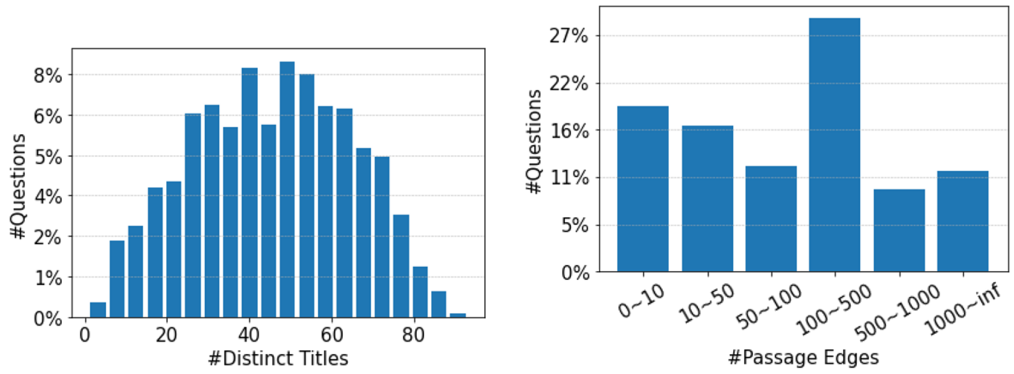
5.3.3 Preliminary Analysis

We conduct preliminary analysis on the graph constructed among passages. Note that for each question, we first apply the retriever to retrieve a few candidate passages, then build edge connection only among the retrieved passages, which means that the passage graph is question-specific. Since the passage graph depends on the retrieved passages, before further utilizing the graph, we need avoid two trivia situations: (1) all the retrieved passages come from the same article; (2) The number of graph edges is very small. Thus we conduct statistics of the passage graphs on two ODQA benchmark datasets, which is shown in Figure 5.2. For each question, the number of retrieved passages is 100. We see that the two trivia situations only happen for a small portion of questions.

5.3.4 Main Results

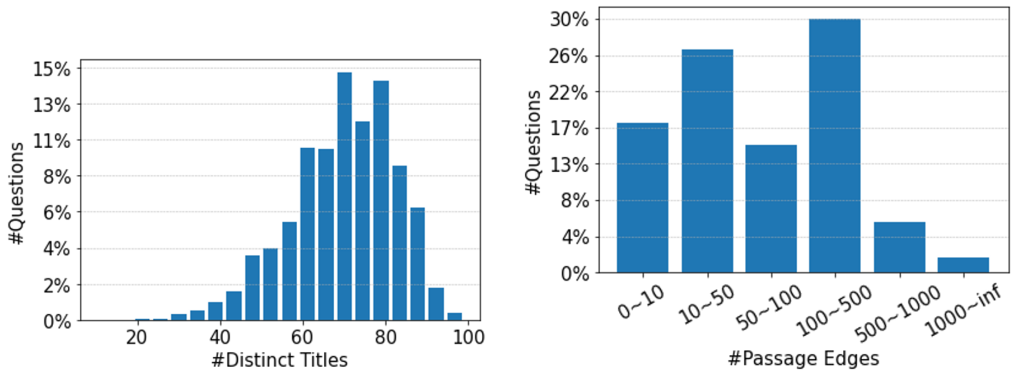
Comparison with Baselines: Table 5.1 shows the results of our method and all baselines. We see that our proposed model KG-FiD consistently and significantly improves FiD on both NQ and TriviaQA datasets over both base and large model. Specifically, for large model, KG-FiD improves FiD by 1.5% and 1.1% on two datasets respectively, which has larger improvement compared to base model. We think the reason is that more expressive reader will also benefit the stage-2 reranking since the initial passage embeddings are generated by the reader encoder module. We also see that our proposed method outperforms all the baseline methods except UniK-QA [68]. However, UniK-QA uses additional knowledge source Wikipedia-Table for retrieval, which is highly related with the NQ dataset and makes it unfair to directly compare with our method.

Efficiency & Accuracy: Table 5.3 show the detailed comparison between our method and FiD in the large model version. The results of base model version is shown in Table 5.2. Besides EM score, we also report the ratio of computation flops (#FLOPs) and inference latency (per question). The detailed calculation of #FLOPs is shown in Appendix 5.3.6. From table 5.3, we see that (1) for KG-FiD, decreasing L_1 can improve the computation efficiency as analyzed in Section 5.2.4, while increasing L_1 can improve the model performance. We think the performance improvement comes from the noise reduction of passage filtering. For a larger L_1, the passage embeddings for reranking will have a better quality so that the gold passages are less likely to be filtered out. (2) Simply reducing the number of passages N_1 into vanilla FiD reader can reduce computation cost, but the performance will also drop significantly (from 51.9 to 50.3 on NQ dataset). (3) Our model can achieve the performance on par with FiD with only 38% of



(a) Results on NQ

(b) Results on TriviaQA



(c) Results on NQ

(d) Results on TriviaQA

Figure 5.2: Preliminary Analysis on the retrieved passages by DPR.

Table 5.1: Exact match score of different models over the test sets of NQ and TriviaQA datasets. $\star$ means that additional knowledge source Wikipedia-Tables is used in this method.

Model	#params	NQ	TriviaQA
T5	11B	36.6	-
GPT-3 (few-shot)	175B	29.9	-
RIDER	626M	48.3	-
RECONSIDER	670M	45.5	61.7
Graph-Retriever	110M	34.7	55.8
Path-Retriever	445M	31.7	-
KAQA	110M	-	66.6
UniK-QA*	990M	**54.0***	64.1*
REALM	330M	40.4	-
RAG	626M	44.5	56.1
Joint Top-K	440M	49.2	64.8
FiD (base)	440M	48.2	65.0
FiD (large)	990M	51.4	67.6
Our Implementation			
FiD (base)	440M	48.8	66.2
KG-FiD (base)	443M	49.6	66.7
FiD (large)	990M	51.9	68.7
KG-FiD (large)	994M	**53.4**	**69.8**

Table 5.2: Inference #FLOPs, Latency (second) and Exact match score of FiD (base) and KG-FiD (base). N_1 is the number of passages into the reader and L_1 is the number of intermediate layers used for stage-2 reranking as introduced in Section 5.2.4. The details of flop computation is introduced in Appendix 5.3.6.

Model	#FLOPs	NQ		TriviaQA	
		EM	Latency (s)	EM	Latency (s)
FiD (N_1=40)	0.40x	47.2	0.27 (0.47x)	64.1	0.27 (0.46x)
FiD (N_1=100)	1.00x	48.8	0.58 (1.00x)	66.2	0.59 (1.00x)
KG-FiD (N_1=100, L_1=3)	0.38x	48.4	0.27 (0.47x)	65.6	0.26 (0.44x)
KG-FiD (N_1=100, L_1=6)	0.56x	49.0	0.35 (0.60x)	66.1	0.34 (0.58x)
KG-FiD (N_1=100, L_1=9)	0.73x	49.3	0.43 (0.74x)	66.3	0.43 (0.73x)
KG-FiD (N_1=100, L_1=12)	0.91x	49.6	0.50 (0.86x)	66.7	0.49 (0.83x)

Table 5.3: Inference #FLOPs, Latency (second) and Exact match score of FiD (large) and KG-FiD (large). N_1 is the number of passages into the reader and L_1 is the number of intermediate layers used for stage-2 reranking as introduced in Section 5.2.4. The details of flop computation is introduced in Appendix 5.3.6.

Model	#FLOPs	NQ		TriviaQA	
		EM	Latency (s)	EM	Latency (s)
FiD (N_1=40)	0.40x	50.3	0.74 (0.45x)	67.5	0.73 (0.44x)
FiD (N_1=100)	1.00x	51.9	1.65 (1.00x)	68.7	1.66 (1.00x)
KG-FiD (N_1=100, L_1=6)	0.38x	52.0	0.70 (0.42x)	68.9	0.68 (0.41x)
KG-FiD (N_1=100, L_1=12)	0.55x	52.3	0.96 (0.58x)	69.2	0.94 (0.57x)
KG-FiD (N_1=100, L_1=18)	0.72x	52.6	1.22 (0.74x)	69.8	1.22 (0.73x)
KG-FiD (N_1=100, L_1=24)	0.90x	53.4	1.49 (0.90x)	69.8	1.48 (0.89x)

Table 5.4: Ablation study of our graph-based reranking method in two stages. EM scores are reported over NQ and Trivia datasets with both base and large model version.

Model	NQ		TriviaQA	
	base	large	base	large
FiD	48.8	51.9	66.2	68.7
KG-FiD	49.6	53.4	66.7	69.8
w/o Stage-1	49.3	53.1	66.2	69.5
w/o Stage-2	49.4	52.3	66.5	69.2

computation cost. When consuming the same amount of computations ($L_1 = 24$), our model significantly outperforms FiD on both NQ and TriviaQA datasets. These experiments demonstrate that our model is very flexible and can improve both the efficiency and effectiveness by changing L_1.

5.3.5 Ablation Study

Effect of Each Reranking Stage: Since our proposed graph-based reranking method are applied in both retrieving stage (Section 5.2.2) and reading stage (Section 5.2.3). We conduct ablation study to validate the effectiveness of each one. Table 5.4 shows the experiment results by removing each module. We see the performance of KG-FiD drops when removing any of the two reranking modules, demonstrating both of them can improve model performance. Another thing we observe is that stage-1 reranking is more effective in base model while stage-2 reranking is more effective in large model. This is reasonable since stage-2 reranking relies on the effectiveness of reader encoder module, where the large model is usually better than the base model.

Passage Ranking Results: We additionally show that our proposed GNN reranking method can improve the passage retrieval results. This is demonstrated in Figure 5.3, where we report Hits@K metric over NQ test set, measuring the percentage of top-K retrieved passages that

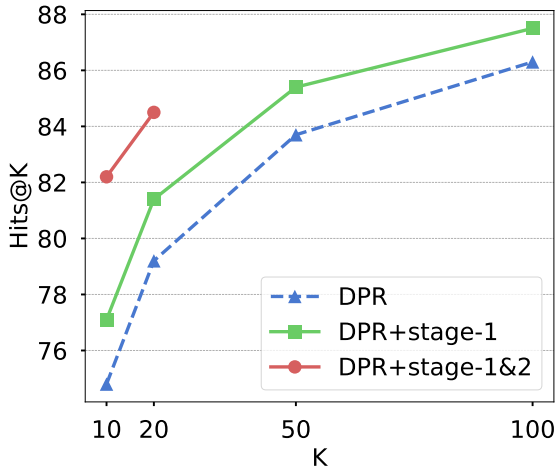


Figure 5.3: Passage ranking results over NQ test set of DPR retriever and our proposed two-stage rerankings over base model.

Table 5.5: Passage Retrieval Results on NQ dev data of our model under different GNN types and number of layers.

Model	H@1	H@5	H@10	H@20
GCN	49.1	69.7	75.7	79.9
GAT	50.1	70.1	76.1	80.2
#Layers				
1	49.0	69.7	75.8	79.8
2	49.6	70.0	76.0	80.2
3	50.1	70.1	76.1	80.2
4	49.5	69.9	76.1	80.1

Table 5.6: EM scores on NQ dev data of our model under different choices of filtered passage numbers and weights of reranking loss.

Model	N_2=10	N_2=20	N_2=30
KG-FiD	47.6	48.0	48.0
	λ=0.01	λ=0.1	λ=1.0
KG-FiD	47.7	48.0	46.6

Table 5.7: #GFLOPs of FiD (base) and KG-FiD (base) over different stages in the model.

Model	Retrieving	Stage-1 Reranking	Reader Encoding	Stage-2 Reranking	Reader Decoding	All
FiD	4.4	-	5772.3	-	714.2	6491.0 (1.00x)
KG-FiD (L_1=3)	4.4	3.5	2308.9	0.4	143.9	2461.1 (0.38x)
KG-FiD (L_1=6)	4.4	3.5	3463.4	0.4	143.9	3615.5 (0.56x)
KG-FiD (L_1=9)	4.4	3.5	4617.9	0.4	143.9	4770.0 (0.73x)
KG-FiD (L_1=12)	4.4	3.5	5772.3	0.4	143.9	5924.5 (0.91x)

contain the gold passages (passages that contain the answer). We see that DPR+stage-1 reranking consistently outperforms DPR for all the $K \in \{10, 20, 50, 100\}$. With two stages of reranking, the retrieval results are further improved for $K \in \{10, 20\}$ (We only cares about $K \leq 20$ for stage-2 reranking since $N_2 = 20$). This shows that such reranking can increase the rank of gold passages which are previously ranked lower by DPR retriever and improve the efficacy of passage pruning.

GNN Model Design: We conduct tuning on the model type and number of layers of our GNN based reranking model. For efficiency, we rerank 100 passages returned by DPR retriever and search them based on the passage retrieval results. Table 5.5 shows the Hits scores for different choices. We see that GAT outperforms vanilla GCN model [54] which is reasonable since GAT leverage attention to reweight neighbor passages by their embeddings. The best choice for the number of GNN layers is 3. Note that other GNN models such as GIN [114], DGI [99] can also be applied here and we leave the further exploration of GNN models as future work.

N_2 **and** λ. For the stage-2 reranking part in Section 5.2.3, we also conduct hyper-parameter search on the number of passages after filtering: $N_2 \in \{10, 20, 30\}$ and the weight of reranking loss when training the reading module: $\lambda \in \{0.01, 0.1, 1.0\}$. As shown in Table 5.6, $N_2 = 20$ achieves better results than $N_2 = 10$, but further increasing N_2 does not bring performance gain while decreasing the efficiency of model since the number of passages to be processed by the decoder is increased. Thus we choose $N_2 = 20$. For the loss weight λ, we found that with its increment, the performance first increases then significantly drops. This shows that it's important to balance the weight of two training losses, as we want the model to learn better passage reranking while not overwhelming the training signal of answer generation.

5.3.6 FLOPs Computation

In this section we compute the FLOPs of each module[4]. The results are shown in Table 5.7 and 5.8 for base model and large model respectively. Before the computation, we first show some basic statistics on two benchmark datasets: the average question length is 20, and the average answer length is 5. For the reading part, the length of concatenated passage question pair is 250, number of input passages is $N_1 = 100$.

We first calculate the number of FLOPs of vanilla FiD model. For the retrieving part, it contains both question encoding and passage similarity search. We only consider the former part

[4]Our computation is based on https://github.com/google-research/electra/blob/master/flops_computation.py

Table 5.8: #GFLOPs of FiD (large) and KG-FiD (large) over different stages in the model.

Model	Retrieving	Stage-1 Reranking	Reader Encoding	Stage-2 Reranking	Reader Decoding	All
FiD	4.4	-	17483.2	-	2534.5	20022.0 (1.00x)
KG-FiD (L_1=6)	4.4	3.5	6993.3	0.6	510.0	7511.8 (0.38x)
KG-FiD (L_1=12)	4.4	3.5	10489.9	0.6	510.0	11008.4 (0.55x)
KG-FiD (L_1=18)	4.4	3.5	13986.5	0.6	510.0	14505.1 (0.72x)
KG-FiD (L_1=24)	4.4	3.5	17483.2	0.6	510.0	18001.7 (0.90x)

as the latter part depends on the corpus size and search methods and is usually very efficient. The question encoding flops by BERT-based model is about 4.4 Gigaflops (GFLOPs). For the reading part, the encoding of each question passage pair takes about 57/174 GFLOPs for base/large model, and the encoding of 100 passages takes 5772/17483 GFLOPs. The decoder part only costs 714.2/2534.5 GFLOPs for base/large model since the average length of answer is very small. In summary, vanilla FiD base/large model costs 6491.0/20022.0 GFLOPs.

For our model, the computation cost of retrieving part is the same as vanilla FiD. Since we set $N_0 = 1000$ and $N_1 = 100$, the GAT [98] computation in stage-1 reranking takes about 3.5 GFLOPs, and the stage-2 reranking takes only 0.4/0.6 GFLOPs for base/large model. For the reader encoding part, the computation cost depends on L_1 and N_2, which is analyzed in Section 5.2.5. For the reader decoding part, where cross attention takes most of the computation, KG-FiD only takes about $N_2/N_1 = 1/5$ cost of vanilla FiD, which is 143.9/510.0 for base/large model respectively. The detailed flops are shown in Table 5.7 and 5.8.

5.4 Summary

This chapter tackles the task of Open-Domain Question Answering. We focus on the current best performed framework FiD and propose a novel KG-based reranking method to enhance the cross-modeling between passages and improve computation efficiency. Our two-stage reranking methods reuses the passage representation generated by DPR retriver and the reader encoder and apply graph neural networks to compute reranking scores. We further propose to use the intermediate layer of encoder to reduce computation cost while still maintaining good performance. Experiments on Natural Questions and TriviaQA show that our model can significantly improve original FiD by 1.5% exact match score and achieve on-par performance with FiD but reducing over 60% of computation cost.

After presenting how to use KGs for question answering when answers are sourced from either KGs or text corpora, we've operated under the assumption that these KGs are complete. However, real-world KGs are often incomplete with missing triplets. Although we've shown that KG completion methods can help mitigate this limitation, a more important question arises: how much can they improve the performance of downstream tasks like question answering? In the upcoming chapter, we will introduce a benchmark designed to thoroughly answer this question.

Chapter 6

Benchmarking the Impacts of KG Completion on Question Answering

6.1 Introduction

As presented in Chapter 3, The inherent incompleteness of Knowledge Graphs (KGs) is a well-recognized problem. As such, a multitude of methods have been proposed to address this issue, spanning statistical relational learning [30], embedding-based models [6, 91, 115], neural-symbolic methods [73, 84, 116], and recent language model-based methods [86, 107, 117]. However, prior studies (including the one in Chapter 3) have largely viewed KG completion as an end in itself, neglecting to investigate its potential impact on subsequent applications that utilize the completed KGs.

On the other hand, Knowledge Graph Question Answering (KGQA), designed to answer natural language questions using information from KGs, is one of the most important applications of KGs. As presented in Chapter 4, extensive KGQA approaches have been proposed in recent years. Despite the rapid advancement, previous studies either operate on complete KGs or handle incompleteness via specific question-answering methods [62, 85, 86], lacking a comprehensive investigation into different approaches. The impact of KGC on KGQA performance remains largely unexplored.

In our study, we seek to rectify this oversight by introducing a novel benchmark CompleQA designed to directly and holistically assess the influence of KG completion on KGQA. This benchmark comprises over three million triplets and approximately $400\,000$ entities across 5 different domains, collectively forming the KG. The corresponding QA dataset includes over 5000 questions, featuring both single-hop and multi-hop questions. These questions are sorted into three generalization levels and are sourced from the GrailQA [33] dataset. For KG completion, we employ entity-centric incompleteness to align with the QA dataset, where the entities in the missing triplets correspond to those appearing in the questions. Importantly, different from previous studies, we actually incorporate predicted triplets into the KG, and use this completed KG for question answering, allowing a seamless study of various KGC and KGQA methods. Our investigation incorporates the study of four representative KG completion models, namely TransE [6], DistMult [115], ComplEx [95], and RotatE [91]. For the KGQA models, we employ two

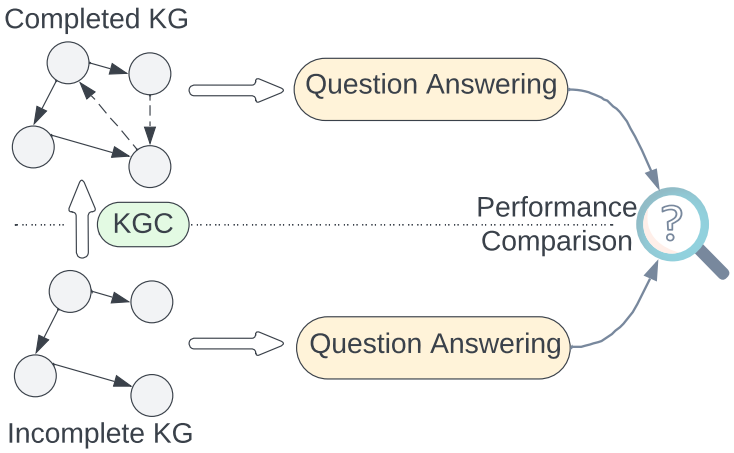


Figure 6.1: We compare the performance between incomplete and completed KGs by examining their question-answering results. Dashed arrows in the completed KG denote new triplets added by KGC.

published leading methods DecAF [124] and Pangu [34].

Our experiments show that the incompleteness of KGs can adversely affect QA performance, and effective KG completion can help alleviate this and improve performance by up to 6.1% across varying degrees of incompleteness. Furthermore, we discern that KG completion performance does not always align with downstream performance, due to the limited overlap in triplet predictions between models and the disparate influence of these triplets on KGC and QA performance. We hope our findings stimulate more research that perceives KG completion not merely as an isolated objective but as a crucial step toward enhancing the performance of downstream tasks.

6.2 Benchmark Construction

In this section, we delineate the procedural steps employed in the development of our benchmark, including the knowledge graph, the question-answering dataset, and the completion dataset.

2.1 Knowledge Graph The choice of the knowledge graph is a fundamental aspect, and in our study, we opt for Freebase [5], primarily owing to its widespread usage in academic circles. The original Freebase contains over 80 million entities. To facilitate affordable exploratory research, we select a subset of Freebase as our final knowledge graph by specifically confining the KG to five important domains: *medicine*, *computer*, *food*, *law*, and *business*. Within this sub-sample of Freebase, we encompass $395\,965$ entities, 383 relations, and $3\,042\,911$ triplets. Despite the reduction in scale, we posit that this knowledge graph retains a substantial breadth and can accurately emulate real-world conditions and challenges. Following this, we explain the construction of the QA and completion datasets derived from this KG.

2.2 Question Answering over KG We denote a QA model intended for $\mathcal{KG}$ as $f_{QA}(\mathcal{KG})$, and follow previous KGQA settings [33, 95, 120], where each natural language question q aligns

Table 6.1: Data Statistics of the QA dataset.

Question Category	Train	Valid	Test
Total	3395	997	973
I.I.D.	-	366	381
Compositional	-	232	225
Zero-Shot	-	399	367

Table 6.2: Data Statistics of the KGC dataset.

Incompleteness	Train	Valid	Test
20%	1 674 405	1807	2864
50%	1 667 397	4519	7160
80%	1 660 390	7230	11 456

with an answer set A and a logical form l, used to extract answers from the KG. We adopt GrailQA [33] as our foundation, which contains both single-hop and multi-hop questions and provides comprehensive evaluation over three generalization difficulty levels: i.i.d., compositional, and zero-shot, determined by whether the KG schemas that appeared in the test set have been observed in the training set. The original GrailQA is based on the full set of Freebase, thus we eliminate questions that cannot be answered or contradict the provided answer labels given our sub-sampled Freebase, This refinement process results in a total of 5365 questions with the statistical details in Table 6.1.

2.3 KG Completion The completion task here refers to the missing entity prediction task: given an incomplete triplet $(e_h, r, ?)$ or $(?, r, e_t)$ where its head or tail entity is missing, the model is required to predict the missing entity[1]. To align it with the QA task for studying its impact, we first identify all the unique entities from the validation and test questions[2], denoted as E_{valid} and E_{test} respectively. We then retrieve all triplets linked to these entities, denoted as $\mathcal{T}'_{valid}$ and $\mathcal{T}'_{test}$. We randomly choose a proportion P of these triplets as the final validation and test sets, $\mathcal{T}_{valid}$ and $\mathcal{T}_{test}$ for KG completion. All remaining triplets, including those unlinked to those entities, form the training data $\mathcal{T}_{train}$. We adjust P to be 20%, 50%, and 80% to introduce varying degrees of incompleteness. The quantity of triplets is detailed in Table 6.2. Note that compared with random sampling from the entire KG, sampling triplets based on entities appeared in the questions can better align the two tasks. Furthermore, this is also seen as a realistic scenario where relatively new entities in KGs often come with incomplete triplets and their related questions are both important and challenging.

2.4 Effect of KGC over KGQA To study the impact of knowledge graph completion on question answering, we incorporate the completed triplet into the KG, then compare the QA performance using the original incomplete KG with the completed KG. This allows us to freely explore any KGC and question-answering methods. The process of incorporating triplets is detailed below.

[1]Following previous conventions, we don't consider the situation of missing classes or literals in this study.
[2]GrailQA provides annotated labels for those entities.

In the case of each incomplete triplet $(e_h, r, ?)$, the model first predicts N candidate tail entities e_{t_i} with scores s_i denoted as $[(e_{t_1}, s_1), \cdots, (e_{t_N}, s_N)]$, where $s_1 \geq \cdots \geq s_N$. To determine whether triplet (e_h, r, e_{t_i}) should be incorporated into the incomplete KG, we establish a threshold s_T. If the score $s_i \geq s_T$, the triplet (e_h, r, e_{t_i}) is added to the KG. Note that we don't add the triplet which is already in the KG. The same process is followed for missing head entity $(?, r, e_t)$. Suppose $\mathcal{T}_{valid}^{pred}$ represents the collection of all added triplets for the validation set and $\mathcal{T}_{test}^{pred}$ for the test set. $\mathcal{KG}(\mathcal{T})$ represents the KG that maintains the consistent set of entities, relations, classes, and literals but incorporates a variable set of triplets $\mathcal{T}$. Finally, we evaluate the performance of the incomplete KG $f_{QA}(\mathcal{KG}(\mathcal{T}_{train}))$ versus the completed KG $f_{QA}(\mathcal{KG}(\mathcal{T}_{train} \cup \mathcal{T}_{valid}^{pred} \cup \mathcal{T}_{test}^{pred}))$. The performance difference can indicate the utility of KG completion over the QA task. Further details on the utilized models, evaluation metrics, and how to determine s_T are provided in the following section.

6.3 Methods and Evaluation Metrics

This section presents the methods implemented on our benchmark and the evaluation metrics employed to assess their effectiveness.

6.3.1 For Question Answering

We employ two state-of-the-art methods, namely DecAF [124] and Pangu [34]. For simplicity, we employ oracle entity linking to Pangu, avoiding the need to train a specially designed entity linker. In contrast, DecAF operates based on text retrieval without the need for entity linking. Due to computational limitations, we chose T5-base [77] as the backbone model for DecAF and BERT-base [23] for Pangu. F1 scores of answer matching are used for the evaluation of QA performance.

6.3.2 For KG Completion

The KGC methods employed in this study comprise four representative models: TransE [6], DistMult [115], ComplEx [95], and RotatE [91]. For the implementation of these methods, we turn to LibKGE [7], a highly developed KGC library. The dimensionality of our embeddings is set to 200, utilizing xavier uniform initialization [31]. Following previous approaches [6, 91], we employ negative sampling for training. In this process, negative triplets are created for every triplet in the training dataset by randomly substituting either the head or tail entity. Each triplet gives rise to 50 negative triplets, half by replacing the head entity and half by substituting the tail. The batch size is 4096, with a maximum of 500 epochs. We deploy early stopping after 5 evaluations over the validation set, with one evaluation after every 10 epochs. When it comes to optimization, we leverage Adam [53] for TransE and RotatE with a learning rate of 0.001. For ComplEx and DistMult, we employ Adagrad [25] with a learning rate within the range [0.01, 1.0]. For RotatE and TransE, we choose from binary cross entropy (BCE), Kullback-Leibler divergence (KL), and margin ranking (MR) for the training losses, and explore the loss argument within [0, 50]. For ComplEx and DistMult, we utilize BCE loss with a loss argument of 0.0. We

Table 6.3: Experiment results on KG completion and question answering of our benchmark CompleQA. Results are averaged across 3 independent runs with different random seeds. "QA w/ X" signifies the QA performance using model X. The top performance in each column of each section is highlighted in **bold**. For QA performance, we provide a percentage change against scenarios where no KG completion was used, which is color-coded: green for positive changes and red for negative ones.

Incompleteness	Model	KGC					QA w/ DecAF	QA w/ Pangu
		MRR	H@1	H@3	H@10	F1	F1	F1
0%	-	-	-	-	-	-	0.782	0.880
20%	None	-	-	-	-	-	0.742	0.846
	TransE	0.614	0.533	0.667	0.761	0.467	0.750 ↑1.1%	0.864 ↑2.1%
	RotatE	0.667	0.612	0.700	**0.763**	0.573	**0.765** ↑3.1%	0.864 ↑2.1%
	DistMult	0.671	0.618	0.718	0.754	0.637	0.757 ↑2.0%	0.865 ↑2.2%
	ComplEx	**0.681**	**0.630**	**0.722**	0.756	**0.660**	0.757 ↑2.0%	**0.868** ↑2.6%
50%	None	-	-	-	-	-	0.672	0.773
	TransE	0.462	0.392	0.500	0.590	0.439	0.684 ↑1.8%	0.765 ↓1.0%
	RotatE	**0.509**	0.455	**0.538**	**0.607**	0.524	0.701 ↑4.3%	0.778 ↑0.6%
	DistMult	0.506	**0.471**	0.528	0.564	0.565	**0.713** ↑6.1%	**0.798** ↑3.2%
	ComplEx	0.504	0.467	0.527	0.564	**0.582**	0.712 ↑6.0%	**0.798** ↑3.2%
80%	None	-	-	-	-	-	0.586	0.638
	TransE	0.299	0.240	0.324	**0.408**	0.335	0.578 ↓1.4%	**0.650** ↑1.9%
	RotatE	**0.312**	**0.265**	**0.328**	0.403	**0.373**	**0.591** ↑0.9%	0.636 ↓0.3%
	DistMult	0.255	0.218	0.270	0.320	0.311	0.585 ↓0.2%	0.640 ↑0.3%
	ComplEx	0.252	0.212	0.272	0.323	0.320	0.588 ↑0.3%	**0.650** ↑1.9%

experimented with other training losses and arguments, but found them ineffective. Hyperparameter search is conducted via Bayesian optimization using the Ax framework (https://ax.dev/), with the number of trials as 40.

To measure the performance of KGC, we adhere to standard metrics for missing entity prediction: Mean Reciprocal Rank (MRR) and Hits@K (H@K) in the filtered setting [6]. Importantly, considering our goal to incorporate predicted triplets into the KG for subsequent question answering, we propose to measure triplet prediction by F1 score: $\text{F1} = 2 \cdot \frac{|\mathcal{T}_{pred} \cap \mathcal{T}_{gold}|}{|\mathcal{T}_{pred}| + |\mathcal{T}_{gold}|}$ where $\mathcal{T}_{pred}$ represents the set of predicted triplets while $\mathcal{T}_{gold}$ denotes the set of missing ground-truth triplets. We adjust the threshold s_T introduced in Section 2.4 on the validation split to achieve the best F1 score. We show such adjustment also produces good QA performance in the following section.

6.4 Experiments

In this section, we delve into the outcomes of our empirical investigations. Firstly, from Table 6.3, it's clear that the KG incompleteness negatively affects performance. Specifically, the F1

Table 6.4: Spearman's rank correlation coefficient between KGC metrics (MRR and F1) and QA metrics (F1) with two models. QA_D denoted the performance of DecAF model while QA_P means the Pangu model.

Incompleteness	Metrics	QA_D-F1	QA_P-F1
20%	KGC-MRR	0.32	0.95
	KGC-F1	0.32	0.95
50%	KGC-MRR	0.40	0.32
	KGC-F1	0.80	0.95
80%	KGC-MRR	0.20	-0.63
	KGC-F1	0.40	-0.30

scores of DecAF and Pangu drop by 14.0% and 12.1% respectively when the KG incompleteness level is 50%. Subsequently, we aim to address the following questions:

Q1: How much does the good performance in KGC translate to the enhancement in QA? The experimental results in Table 6.3 demonstrate that KGC can indeed enhance the QA performance in the majority of the cases. Notably, the ComplEx algorithm for KGC boosted the QA performance of DecAF alone (without KGC) by 6.0% and of Pangu by 3.2% at 50% incompleteness level. However, at the 80% incompleteness level, the same ComlEx algorithm only boosted the QA performance of DecAF by 0.3% and of Pangu by 1.9%, which are much less than the performance gains at the 50% and 20% incompleteness levels. The observations on other KGC methods demonstrated similar patterns; some of them even lead to decreased QA results. This could be attributed to that incorrect triplets introduced by the KGC methods outweigh the correctly predicted ones.

To validate this, we construct a scenario that incorporates only the correctly-predicted triplets into the KG, while all the incorrect triplets are discarded. Figure 6.2 clearly illustrates a significant performance enhancement in this scenario, especially at the 80% incompleteness level, thereby substantiating the detrimental impact of incorrect predicted triplets on the QA model.

Q2: Does the best KGC method(s) always lead to the best QA performance? According to Table 6.3, we see that better KGC does not always translate to better downstream outcomes. As an illustration, although ComplEx achieves exceptional KGC results at 20% incompleteness, it does not outperform RotatE in terms of QA performance when DecAF is used. Similarly, even though RotatE produces top-tier KGC outcomes at 80% incompleteness, it even leads to the worst QA performance when Pangu is utilized.

We use Spearman's rank correlation coefficient to quantitatively assess the relationship between the performance of KGC and QA. This coefficient measures the Pearson correlation between the ranked values of two variables. In many instances, we find that the correlation between the KGC metric and QA metric is weak, with a value below 0.5. Another observation is that the F1 score of KGC performance corresponds better with QA performance than the mean reciprocal rank (MRR). This is reasonable because QA performance relies on the completed knowledge graph, which includes predicted triplets. The F1 score, in this context, is a more direct measure of the quality of predicted triplets compared to MRR.

To delve further, we measured the overlap of predicted triplets among various models and

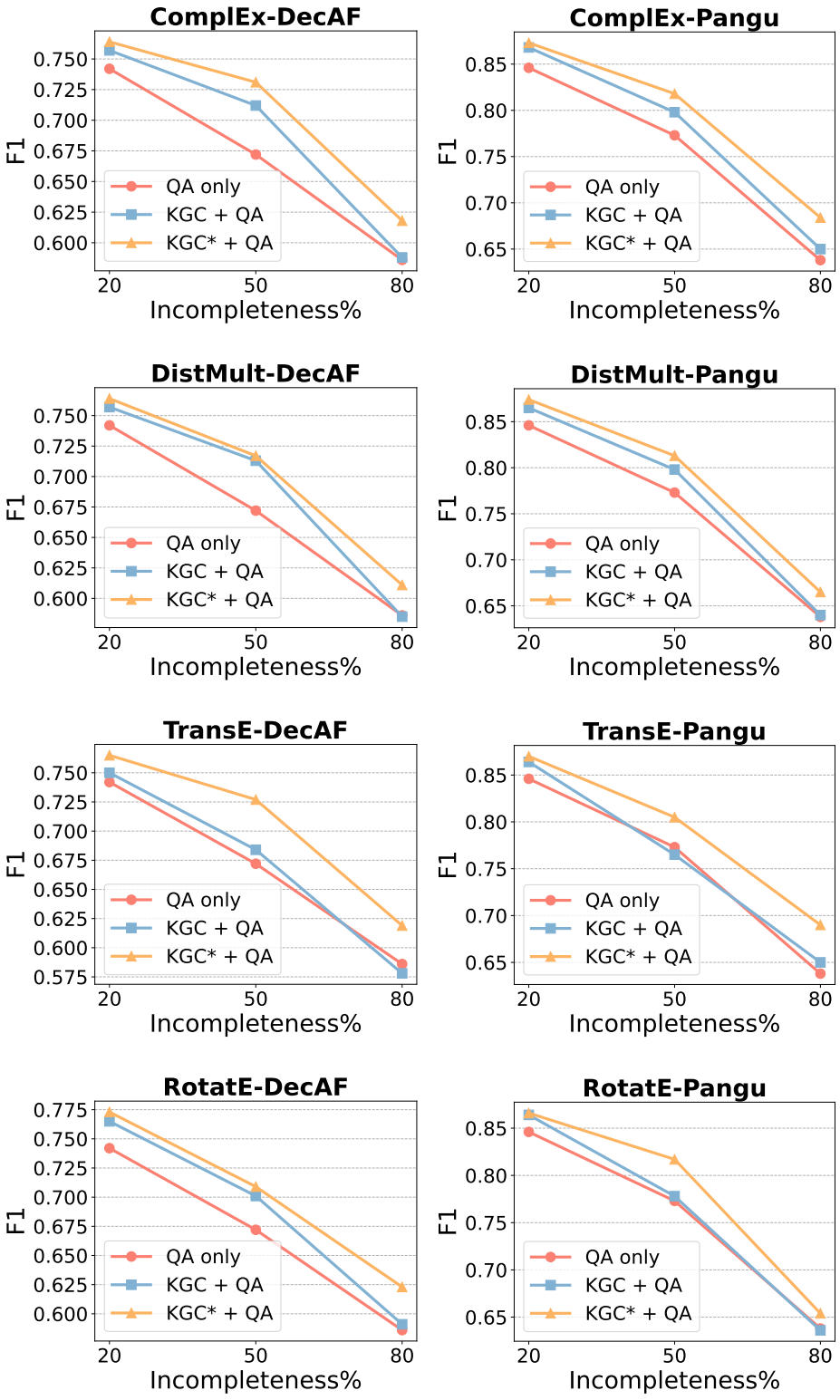


Figure 6.2: QA performance under different completed KGs. "QA only" means using the original incomplete KG for QA. "KGC + QA" means using the KGC-completed KG. KGC* means only keeping the correct triplets while discarding all the incorrect ones.

found that no model's predictions, correct or incorrect, fully encompassed the other's. For example, in a 20% incomplete KG, ComplEx and RotatE had about 80% shared correct predictions and 30% shared incorrect ones. In this case, despite ComplEx's superior performance in KGC, it doesn't solely determine QA performance as various predicted triplets impact QA differently, and this impact may not align well with their contribution to KGC performance. This discrepancy points to the need for KGC methods that optimize both KG completion and downstream task performance.

Q3: How does the score threshold affect the KGC and QA performance? We aimed to study the effect of score threshold s_T for each KGC method, which is introduced in Section 2.4. We vary this threshold for each method to evaluate the corresponding KGC and QA performance. As shown in Figure 6.3, the KGC F1 score initially rises and then falls, as increasing the score threshold leads to fewer triplets being added, which although improves precision, it negatively impacts recall. We see that the curve representing the relationship between the score threshold and the KGC F1 score appears to coincide well with the QA performance curve. However, there are instances where this alignment is less pronounced, indicating that utilizing KGC F1 to pinpoint the task-specific score threshold provides a useful starting point, but is not sufficient on its own. We believe that further investigation of this issue could provide valuable insights for future research.

6.5 Summary

In this chapter, we introduced a novel benchmark to investigate the impact of representative KGC methods on the Knowledge Graph Question Answering (KGQA) task. Our findings demonstrate that KG incompleteness negatively affects KGQA, and effective KGC can significantly mitigate this issue. However, we also discovered that best-performing KGC method does not necessarily lead to the best KGQA results. Our work underlines the necessity to view KGC not merely as a standalone goal, but as a vital step toward improving downstream tasks. Furthermore, we hope this chapter can facilitate the joint study of KG acquisition and application in future research, as opposed to studying them separately.

Finally, in the concluding chapter, we will synthesize the key findings and propose potential avenues for further research grounded in our acquired knowledge.

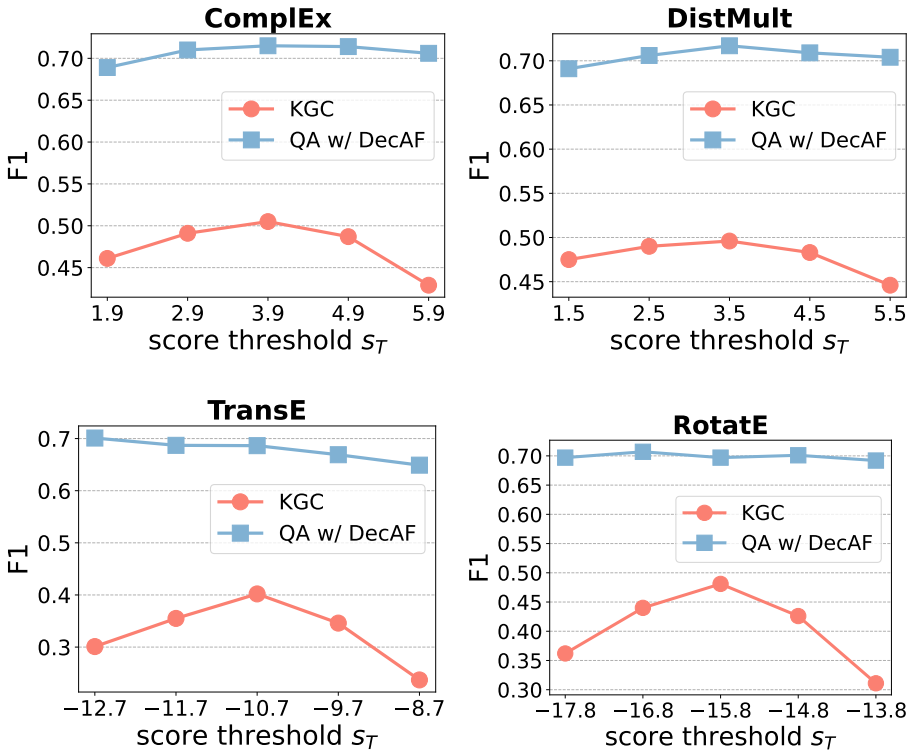


Figure 6.3: Performance variation with different score threshold. Each plot depicts the performance trends for one KGC method at 50% KG incompleteness level. The curves represent changes in KGC F1 scores on the validation split and QA w/ DecAF F1 scores on the test split as the score threshold varies.

Chapter 7

Concluding Remarks and Future Work

This thesis aims to effectively leverage textual semantics for KG acquisition and application, with a primary focus on Pre-trained Language Models (PLMs). This goal is achieved through two key dimensions, and the related contributions are summarized as follows:

- **KG Acquisition Through Text:** We started our exploration with a KG representation learning framework called JAKET in Chapter 2. This framework leverages joint pre-training to optimize both KG and text embeddings simultaneously, enhancing each other in the process. These optimized representations have shown outstanding results in KG acquisition tasks including relation extraction and entity classification. To address the challenges of scaling to large knowledge graphs and performing KG completion, we introduce ResKGC in Chapter 3. This seq2seq model takes a text-based incomplete triplet as input and returns the name of the missing entity as output. We propose a text-based retrieval method to identify relevant triplets to the incomplete one, subsequently improving the accuracy of missing entity prediction.
- **Answering Text Question with KGs:** We next explored how to leverage KGs for question-answering applications. Firstly, we created DecAF, as presented in Chapter 4, a framework capable of generating both logical queries and direct answers simultaneously. This addresses the issue of non-executable logical queries found in previous methods. We then developed KG-FiD in Chapter 5, which improves open-domain question answering by incorporating a KG-based passage reranking mechanism. We demonstrated that even when answers are sourced from text corpora, KGs can still be utilized to enhance the QA performance. Lastly, we evaluated the influence of KG completion on question-answering performance by proposing a new benchmark, CompleQA in Chapter 6. Our findings indicate that while KG completion can improve downstream QA performance, the best-performing KG completion methods do not necessarily yield the best QA results. This underscores the importance of jointly studying KG acquisition and applications.

We provide a summary of this thesis at the methodological level on how to effectively leverage textual semantics in the context of structural KGs. This is illustrated through two distinct approaches, as detailed below.

- **Hybrid Model:** This involves the integration of Graph Neural Networks (GNNs) with PLMs. In this approach, each type of model focuses on its area of expertise—PLMs on

textual data and GNNs on graph structures. For instance, in the JAKET framework (Chapter 2), the PLM initially computes entity representations that are then fed into the GNN for further processing. Conversely, the GNN outputs refined entity representations back into the PLM. In the case of KG-FiD (Chapter 5), GNNs utilize the PLM-output embeddings as initial inputs, which are then employed for passage reranking.

- **KG Linearization:** This approach converts the structural KG into a linear text sequence, thereby allowing the exclusive use of PLMs for processing. In frameworks like ResKGC (Chapter 3) and DecAF (Chapter 4), a KG triplet is linearized into a sentence by concatenating the names of the head entity, relation, and the tail entity. These sentences are then grouped into passages based on the head entity.

Each approach has its own advantages and disadvantages: The Hybrid Model offers explicit structural modeling through GNNs, but this complicates the joint optimization process with PLMs. Conversely, the KG Linearization simplifies the optimization by relying solely on PLMs, but converting KGs to text leads to a loss of structural information. For future exploration, the former should aim for more effective integration of GNNs and PLMs, while the latter should work on improving KG linearization methods to preserve more structural information.

Recognizing the foundational importance of PLMs in our studies, and observing the transformative impact of evolving large language models (LLMs), the future implications of KGs in the LLM era are worth exploring. LLMs, despite their many strengths, have identifiable limitations. A major one is the occurrence of hallucinations, where the model produces factually incorrect information. For instance, if one were to ask GPT-4 [69], "How many main bosses does Dark Souls I have?", it may incorrectly reply "11" when the accurate answer is "13".

How can KGs mitigate such hallucinations? During the training phase, KGs can serve as additional data to improve the pre-training process so that LLMs can store more factual knowledge, similar to our work JAKET (Chapter 2) which pre-trains on KG and text jointly. In the inference stage, relevant information from KGs can be retrieved to supplement the input queries and increase response accuracy. Our DecAF (Chapter 4) framework can be regarded as an example where LLMs can serve as the reader module.

Conversely, KGs have their own limitations, such as a restricted domain scope due to the challenges of constructing them. For instance, a KG focused primarily on finance may be ill-suited for answering medical queries. To address this, targeted and efficient KG construction is essential. LLMs have great potential to facilitate this process by automatically generating or refining KGs. Our ResKGC (Chapter 3), which uses PLMs for sequence-to-sequence KG completion, suggests that LLMs can take this role effectively.

In summary, our thesis stands as a meaningful contribution to leverage textual semantics for both KG acquisition and application. We also anticipate that it will offer valuable perspectives on the future role of KGs in the LLM landscape.

Bibliography

[1] Akari Asai, Kazuma Hashimoto, Hannaneh Hajishirzi, Richard Socher, and Caiming Xiong. Learning to retrieve reasoning paths over wikipedia graph for question answering. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=SJgVHkrYDH. 5.3.2

[2] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. Dbpedia: A nucleus for a web of open data. In *The semantic web*, pages 722–735. Springer, 2007. 4.1

[3] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016. 2.2.2, 2.2.4

[4] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1247–1250, 2008. 3.1

[5] Kurt D. Bollacker, Colin Evans, Praveen K. Paritosh, Tim Sturge, and Jamie Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *SIGMOD Conference*, 2008. 4.1, 4.3, 6.2

[6] Antoine Bordes, Nicolas Usunier, Alberto García-Durán, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. 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 2787–2795, 2013. URL https://proceedings.neurips.cc/paper/2013/hash/1cecc7a77928ca8133fa24680a88d2f9-Abstract.html. 2.1, 2.2.2, 3.1, 3.4.1, 3.2, 3.3, 6.1, 6.1, 6.3.2

[7] Samuel Broscheit, Daniel Ruffinelli, Adrian Kochsiek, Patrick Betz, and Rainer Gemulla. LibKGE - a knowledge graph embedding library for reproducible research. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Association for Computational Linguistics, 2020. doi: 10.18653/v1/2020.emnlp-demos.22. URL https://aclanthology.org/2020.emnlp-demos.22. 6.3.2

[8] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html. 5.3.2

[9] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020. 1.1, 2.1

[10] Shulin Cao, Jiaxin Shi, Zijun Yao, Xin Lv, Jifan Yu, Lei Hou, Juanzi Li, Zhiyuan Liu, and Jinghui Xiao. Program transfer for answering complex questions over knowledge bases. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8128–8140, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-long.559. URL https://aclanthology.org/2022.acl-long.559. 4.3

[11] Linlin Chao, Taifeng Wang, and Wei Chu. Pie: a parameter and inference efficient solution for large scale knowledge graph embedding reasoning. *arXiv preprint arXiv:2204.13957*, 2022. 3.3

[12] Chen Chen, Yufei Wang, Bing Li, and Kwok-Yan Lam. Knowledge is flat: A Seq2Seq generative framework for various knowledge graph completion. In *Proceedings of the 29th International Conference on Computational Linguistics*, pages 4005–4017, Gyeongju, Republic of Korea, October 2022. International Committee on Computational Linguistics. URL https://aclanthology.org/2022.coling-1.352. 3.1, 3.2, 3.3.3

[13] Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. Reading wikipedia to answer open-domain questions. In *ACL*, 2017. 4.2.1

[14] Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. Reading Wikipedia to answer open-domain questions. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1870–1879, Vancouver, Canada, 2017. Association for Computational Linguistics. doi: 10.18653/v1/P17-1171. URL https://aclanthology.org/P17-1171. 5.1

[15] Shuang Chen, Qian Liu, Zhiwei Yu, Chin-Yew Lin, Jian-Guang Lou, and Feng Jiang. Retrack: A flexible and efficient framework for knowledge base question answering. In *ACL*, 2021. 4.1, 4.3, 4.3

[16] Yongrui Chen, Huiying Li, Guilin Qi, Tianxing Wu, and Tenggou Wang. Outlining and filling: Hierarchical query graph generation for answering complex questions over knowl-

edge graph. *ArXiv*, abs/2111.00732, 2021. 4.3

[17] Louis Clouatre, Philippe Trempe, Amal Zouaq, and Sarath Chandar. MLMLM: Link prediction with mean likelihood masked language model. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 4321–4331, Online, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.findings-acl.378. URL https://aclanthology.org/2021.findings-acl.378. 3.2

[18] Rajarshi Das, Manzil Zaheer, Dung Ngoc Thai, Ameya Godbole, Ethan Perez, Jay Yoon Lee, Lizhen Tan, Lazaros Polymenakos, and Andrew McCallum. Case-based reasoning for natural language queries over knowledge bases. In *EMNLP*, 2021. 4.1, 4.3, 4.3

[19] Rajarshi Das, Ameya Godbole, Ankita Rajaram Naik, Elliot Tower, Robin Jia, Manzil Zaheer, Hannaneh Hajishirzi, and Andrew McCallum. Knowledge base question answering by case-based reasoning over subgraphs. In *ICML*, 2022. 4.3

[20] Nicola De Cao, Gautier Izacard, Sebastian Riedel, and Fabio Petroni. Autoregressive entity retrieval. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL https://openreview.net/forum?id=5k8F6UU39V. 3.2

[21] Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. Convolutional 2d knowledge graph embeddings. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018. 2.1, 3.1

[22] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL https://www.aclweb.org/anthology/N19-1423. 1.1, 2.1, 2.2.1, 2.2.3, 2.1

[23] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL https://aclanthology.org/N19-1423. 4.2.2, 4.3, 6.3.1

[24] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL https://aclanthology.org/N19-1423. 5.1

[25] John C. Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. In Adam Tauman Kalai and Mehryar Mohri,

editors, *COLT 2010 - The 23rd Conference on Learning Theory, Haifa, Israel, June 27-29, 2010*. Omnipress, 2010. URL http://colt2010.haifa.il.ibm.com/papers/COLT2010proceedings.pdf#page=265. 6.3.2

[26] Sarah Elhammadi, Laks VS Lakshmanan, Raymond Ng, Michael Simpson, Baoxing Huai, Zhefeng Wang, and Lanjun Wang. A high precision pipeline for financial knowledge graph construction. In *Proceedings of the 28th international conference on computational linguistics*, pages 967–977, 2020. 1

[27] Anna Fensel, Zaenal Akbar, Elias Kärle, Christoph Blank, Patrick Pixner, and Andreas Gruber. Knowledge graphs for online marketing and sales of touristic services. *Information*, 11(5):253, 2020. 1

[28] Thibault Févry, Livio Baldini Soares, Nicholas FitzGerald, Eunsol Choi, and Tom Kwiatkowski. Entities as experts: Sparse memory access with entity supervision. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4937–4951, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.400. URL https://www.aclweb.org/anthology/2020.emnlp-main.400. 2.1

[29] Tianyu Gao, Xu Han, Hao Zhu, Zhiyuan Liu, Peng Li, Maosong Sun, and Jie Zhou. FewRel 2.0: Towards more challenging few-shot relation classification. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 6250–6255, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1649. URL https://www.aclweb.org/anthology/D19-1649. (document), 2.3.2, 2.1

[30] Lise Getoor and Ben Taskar. *Introduction to statistical relational learning*. MIT press, 2007. 6.1

[31] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, 2010. 6.3.2

[32] Yu Gu and Yu Su. Arcaneqa: Dynamic program induction and contextualized encoding for knowledge base question answering. *ArXiv*, abs/2204.08109, 2022. 4.3, 4.3

[33] Yu Gu, Sue Kase, Michelle Vanni, Brian Sadler, Percy Liang, Xifeng Yan, and Yu Su. Beyond iid: three levels of generalization for question answering on knowledge bases. In *Proceedings of the Web Conference 2021*, pages 3477–3488, 2021. 4.1, 4.3, 6.1, 6.2

[34] Yu Gu, Xiang Deng, and Yu Su. Don't generate, discriminate: A proposal for grounding language models to real-world environments. *ArXiv preprint*, abs/2212.09736, 2022. URL https://arxiv.org/abs/2212.09736. 6.1, 6.3.1

[35] Yu Gu, Vardaan Pahuja, Gong Cheng, and Yu Su. Knowledge base question answering: A semantic parsing perspective. *arXiv preprint arXiv:2209.04994*, 2022. 4.1

[36] Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. Realm:

Retrieval-augmented language model pre-training. *arXiv preprint arXiv:2002.08909*, 2020. 5.1, 5.3.2

[37] William L. Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. 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 1024–1034, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/5dd9db5e033da9c6fb5ba83c7a7ebea9-Abstract.html. 2.2.2

[38] Xu Han, Hao Zhu, Pengfei Yu, Ziyun Wang, Yuan Yao, Zhiyuan Liu, and Maosong Sun. FewRel: A large-scale supervised few-shot relation classification dataset with state-of-the-art evaluation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4803–4809, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1514. URL https://www.aclweb.org/anthology/D18-1514. 2.3.2

[39] Gaole He, Yunshi Lan, Jing Jiang, Wayne Xin Zhao, and Ji rong Wen. Improving multi-hop knowledge base question answering by learning intermediate supervision signals. *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*, 2021. 4.3

[40] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016. 2.2.6

[41] E. Prud hommeaux. Sparql query language for rdf. 2011. 4.1

[42] Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay S. Pande, and Jure Leskovec. Strategies for pre-training graph neural networks. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=HJlWWJSFDH. 2.2.6

[43] Weihua Hu, Matthias Fey, Hongyu Ren, Maho Nakata, Yuxiao Dong, and Jure Leskovec. Ogb-lsc: A large-scale challenge for machine learning on graphs. *arXiv preprint arXiv:2103.09430*, 2021. (document), 3.1, 3.4.1, 3.3

[44] Srinivasan Iyer, Sewon Min, Yashar Mehdad, and Wen-tau Yih. RECONSIDER: Improved re-ranking using span-focused cross-attention for open domain question answering. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1280–1287, Online, 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.100. URL https://aclanthology.org/2021.naacl-main.100. 5.3.2

[45] Gautier Izacard and Edouard Grave. Distilling knowledge from reader to retriever for question answering. *arXiv preprint arXiv:2012.04584*, 2020. URL https://arxiv.org/abs/2012.04584. 5.1

[46] Gautier Izacard and Edouard Grave. Leveraging passage retrieval with generative mod-

els for open domain question answering. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 874–880, Online, 2021. Association for Computational Linguistics. URL https://aclanthology.org/2021.eacl-main.74. 1.1.2, 3.3.3, 4.2.1, 4.2.3, 4.3, 5.1, 5.3, 5.3.1, 5.3.2

[47] Kelvin Jiang, Dekun Wu, and Hui Jiang. Freebaseqa: A new factoid qa data set matching trivia-style question-answer pairs with freebase. In *NAACL*, 2019. 4.1, 4.3

[48] Zhengbao Jiang, Jun Araki, Donghan Yu, Ruohong Zhang, Wei Xu, Yiming Yang, and Graham Neubig. Learning relation entailment with structured and textual information. In *Automated Knowledge Base Construction*, 2020. 1.3

[49] Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with gpus. *IEEE Transactions on Big Data*, 2019. 4.2.2, 5.2.2

[50] Mandar Joshi, Eunsol Choi, Daniel Weld, and Luke Zettlemoyer. TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1601–1611, Vancouver, Canada, 2017. Association for Computational Linguistics. doi: 10.18653/v1/P17-1147. URL https://aclanthology.org/P17-1147. 5.3

[51] Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6769–6781, Online, 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.550. URL https://aclanthology.org/2020.emnlp-main.550. 3.3.2, 4.2.1, 4.2.2, 4.3, 4.3.1, 5.1, 5.2.1, 5.2.2, 2, 5.3.1

[52] Seyed Mehran Kazemi and David Poole. Simple embedding for link prediction in knowledge graphs. *Advances in neural information processing systems*, 31, 2018. 3.2

[53] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6980. 3.4.1, 6.3.2

[54] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=SJU4ayYgl. 2, 5.3.5

[55] Adrian Kochsiek and Rainer Gemulla. Parallel training of knowledge graph embedding models: a comparison of techniques. *Proceedings of the VLDB Endowment*, 15(3):633–645, 2021. (document), 3.2

[56] Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina

Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. Natural questions: A benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7: 452–466, 2019. doi: 10.1162/tacl_a_00276. URL https://aclanthology.org/Q19-1026. 5.3

[57] Yunshi Lan and Jing Jiang. Query graph generation for answering multi-hop complex questions from knowledge bases. In *ACL*, 2020. 4.3, 4.3

[58] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.703. URL https://aclanthology.org/2020.acl-main.703. 4.2.3

[59] Patrick S. H. Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/6b493230205f780e1bc26945df7481e5-Abstract.html. 5.1, 5.3.2

[60] Belinda Z. Li, Sewon Min, Srinivasan Iyer, Yashar Mehdad, and Wen-tau Yih. Efficient one-pass end-to-end entity linking for questions. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6433–6441, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.522. URL https://aclanthology.org/2020.emnlp-main.522. 4.1

[61] Jimmy J. Lin, Xueguang Ma, Sheng-Chieh Lin, Jheng-Hong Yang, Ronak Pradeep, Rodrigo Nogueira, and David R. Cheriton. Pyserini: A python toolkit for reproducible information retrieval research with sparse and dense representations. *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2021. 4.3

[62] Lihui Liu, Boxin Du, Jiejun Xu, Yinglong Xia, and Hanghang Tong. Joint knowledge graph completion and question answering. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2022. 6.1

[63] 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. *arXiv preprint arXiv:1907.11692*, 2019. 2.2.3, 2.3.1, 2.1

[64] Zhenghao Liu, Chenyan Xiong, Maosong Sun, and Zhiyuan Liu. Entity-duet neural

ranking: Understanding the role of knowledge graph semantics in neural information retrieval. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2395–2405, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-1223. URL https://aclanthology.org/P18-1223. 1

[65] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7. 2.3.1, 5.3.1

[66] Yuning Mao, Pengcheng He, Xiaodong Liu, Yelong Shen, Jianfeng Gao, Jiawei Han, and Weizhu Chen. Reader-guided passage reranking for open-domain question answering. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 344–350, Online, 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.findings-acl.29. URL https://aclanthology.org/2021.findings-acl.29. 5.3.2

[67] Sewon Min, Danqi Chen, Luke Zettlemoyer, and Hannaneh Hajishirzi. Knowledge guided text retrieval and reading for open domain question answering. *arXiv preprint arXiv:1911.03868*, 2019. URL https://arxiv.org/abs/1911.03868. 5.1, 5.2.1, 5.3.2

[68] Barlas Oguz, Xilun Chen, Vladimir Karpukhin, Stan Peshterliev, Dmytro Okhonko, Michael Schlichtkrull, Sonal Gupta, Yashar Mehdad, and Scott Yih. Unified open-domain question answering with structured and unstructured knowledge. *arXiv preprint arXiv:2012.14610*, 2020. URL https://arxiv.org/abs/2012.14610. 5.3.2, 5.3.4

[69] R OpenAI. Gpt-4 technical report. *arXiv*, pages 2303–08774, 2023. 7

[70] Barlas Oğuz, Xilun Chen, Vladimir Karpukhin, Stanislav Peshterliev, Dmytro Okhonko, M. Schlichtkrull, Sonal Gupta, Yashar Mehdad, and Scott Yih. Unik-qa: Unified representations of structured and unstructured knowledge for open-domain question answering. In *NAACL-HLT*, 2022. 4.1, 4.3, 4.3

[71] Matthew E. Peters, Mark Neumann, Robert Logan, Roy Schwartz, Vidur Joshi, Sameer Singh, and Noah A. Smith. Knowledge enhanced contextual word representations. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 43–54, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1005. URL https://www.aclweb.org/anthology/D19-1005. 2.1, 2.3.1, 2.1

[72] Peng Qi, Haejun Lee, Tg Sido, and Christopher Manning. Answering open-domain questions of varying reasoning steps from text. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3599–3614, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.292. URL https://aclanthology.org/

2021.emnlp-main.292. 4.3.2

[73] Meng Qu, Junkun Chen, Louis-Pascal A. C. Xhonneux, Yoshua Bengio, and Jian Tang. Rnnlogic: Learning logic rules for reasoning on knowledge graphs. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL https://openreview.net/forum?id=tGZu6DlbreV. 6.1

[74] Yingqi Qu, Yuchen Ding, Jing Liu, Kai Liu, Ruiyang Ren, Wayne Xin Zhao, Daxiang Dong, Hua Wu, and Haifeng Wang. RocketQA: An optimized training approach to dense passage retrieval for open-domain question answering. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 5835–5847, Online, 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.466. URL https://aclanthology.org/2021.naacl-main.466. 5.1

[75] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. *https://s3-us-west-2.amazonaws.com/openai-assets/research-covers/language-unsupervised/language_understanding_paper.pdf*, 2018. 2.1

[76] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8):9, 2019. 2.1

[77] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*, 2019. URL https://arxiv.org/abs/1910.10683. 1.1, 2.1, 3.2, 3.3.3, 3.4.1, 4.2.3, 4.2.4, 4.3, 5.2.3, 5.3.1, 6.3.1

[78] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas, 2016. Association for Computational Linguistics. doi: 10.18653/v1/D16-1264. URL https://aclanthology.org/D16-1264. 5.3.1

[79] Michael Ringgaard, Rahul Gupta, and Fernando CN Pereira. Sling: A framework for frame semantic parsing. *arXiv preprint arXiv:1710.07032*, 2017. 2.3.1

[80] Adam Roberts, Colin Raffel, and Noam Shazeer. How much knowledge can you pack into the parameters of a language model? In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 5418–5426, Online, 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.437. URL https://aclanthology.org/2020.emnlp-main.437. 5.1, 5.3.2

[81] Stephen Robertson, Hugo Zaragoza, et al. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends® in Information Retrieval*, 3(4):333–389, 2009. 3.1, 3.3.2, 4.2.2

[82] Maya Rotmensch, Yoni Halpern, Abdulhakim Tlimat, Steven Horng, and David Sontag. Learning a health knowledge graph from electronic medical records. *Scientific reports*, 7

(1):1–11, 2017. 1

[83] Devendra Sachan, Mostofa Patwary, Mohammad Shoeybi, Neel Kant, Wei Ping, William L. Hamilton, and Bryan Catanzaro. End-to-end training of neural retrievers for open-domain question answering. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 6648–6662, Online, 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.519. URL https://aclanthology.org/2021.acl-long.519. 5.3.2

[84] Ali Sadeghian, Mohammadreza Armandpour, Patrick Ding, and Daisy Zhe Wang. Drum: End-to-end differentiable rule mining on knowledge graphs. *Advances in Neural Information Processing Systems*, 32, 2019. 6.1

[85] Apoorv Saxena, Aditay Tripathi, and Partha Pratim Talukdar. Improving multi-hop question answering over knowledge graphs using knowledge base embeddings. In *ACL*, 2020. 6.1

[86] Apoorv Saxena, Adrian Kochsiek, and Rainer Gemulla. Sequence-to-sequence knowledge graph completion and question answering. In *ACL*, 2022. (document), 3.1, 3.2, 3.3, 4.1, 4.3, 6.1

[87] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *European Semantic Web Conference*, pages 593–607. Springer, 2018. 2.1

[88] Haitian Sun, Tania Bedrax-Weiss, and William W. Cohen. Pullnet: Open domain question answering with iterative retrieval on knowledge bases and text. *ArXiv*, abs/1904.09537, 2019. 4.1, 4.3

[89] Haitian Sun, Andrew O. Arnold, Tania Bedrax-Weiss, Fernando Pereira, and William W. Cohen. Faithful embeddings for knowledge base queries. *arXiv: Learning*, 2020. 4.3

[90] Tianxiang Sun, Yunfan Shao, Xipeng Qiu, Qipeng Guo, Yaru Hu, Xuanjing Huang, and Zheng Zhang. Colake: Contextualized language and knowledge embedding. *arXiv preprint arXiv:2010.00309*, 2020. 2.3.1

[91] Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, and Jian Tang. Rotate: Knowledge graph embedding by relational rotation in complex space. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=HkgEQnRqYQ. 2.1, 3.2, 6.1, 6.1, 6.3.2

[92] Zhiqing Sun, Hongkun Yu, Xiaodan Song, Renjie Liu, Yiming Yang, and Denny Zhou. Mobilebert: a compact task-agnostic bert for resource-limited devices. *arXiv preprint arXiv:2004.02984*, 2020. 2.3.3

[93] Alon Talmor and Jonathan Berant. The web as a knowledge-base for answering complex questions. In *NAACL*, 2018. 4.1, 4.3

[94] Kristina Toutanova and Danqi Chen. Observed versus latent features for knowledge base and text inference. In *Proceedings of the 3rd Workshop on Continuous Vector*

Space Models and their Compositionality, pages 57–66, Beijing, China, July 2015. Association for Computational Linguistics. doi: 10.18653/v1/W15-4007. URL https://aclanthology.org/W15-4007. 3.1

[95] Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. Complex embeddings for simple link prediction. In Maria-Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 2071–2080. JMLR.org, 2016. URL http://proceedings.mlr.press/v48/trouillon16.html. 3.2, 3.3, 6.1, 6.2, 6.3.2

[96] Shikhar Vashishth, Soumya Sanyal, Vikram Nitin, and Partha P. Talukdar. Composition-based multi-relational graph convolutional networks. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=BylA_C4tPr. 2.2.2

[97] 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. URL https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html. 1.1, 2.1, 3.1

[98] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL https://openreview.net/forum?id=rJXMpikCZ. 2.1, 2.2.2, 5.2.2, 5.2.3, 5.3.1, 5.3.6

[99] Petar Velickovic, William Fedus, William L. Hamilton, Pietro Liò, Yoshua Bengio, and R. Devon Hjelm. Deep graph infomax. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=rklz9iAcKQ. 5.3.5

[100] Pat Verga, Haitian Sun, Livio Baldini Soares, and William W Cohen. Facts as experts: Adaptable and interpretable neural memory over symbolic knowledge. *arXiv preprint arXiv:2007.00849*, 2020. 2.1

[101] Pat Verga, Haitian Sun, Livio Baldini Soares, and William Cohen. Adaptable and interpretable neural memoryover symbolic knowledge. In *NAACL*, 2021. 4.3

[102] Denny Vrandečić and Markus Krötzsch. Wikidata: a free collaborative knowledgebase. *Communications of the ACM*, 57(10):78–85, 2014. 3.1, 4.1, 5.2.1

[103] Denny Vrandečić and Markus Krötzsch. Wikidata: a free collaborative knowledgebase. *Communications of the ACM*, 57(10):78–85, 2014. 2.3.1

[104] Hongwei Wang, Fuzheng Zhang, Miao Zhao, Wenjie Li, Xing Xie, and Minyi Guo. Multi-task feature learning for knowledge graph enhanced recommendation. In *The world wide web conference*, pages 2000–2010, 2019. 1

[105] Liang Wang, Wei Zhao, Zhuoyu Wei, and Jingming Liu. SimKGC: Simple contrastive knowledge graph completion with pre-trained language models. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 4281–4294, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-long.295. URL https://aclanthology.org/2022.acl-long.295. 3.2

[106] Minjie Wang, Lingfan Yu, Da Zheng, Quan Gan, Yu Gai, Zihao Ye, Mufei Li, Jinjing Zhou, Qi Huang, Chao Ma, et al. Deep graph library: Towards efficient and scalable deep learning on graphs. *arXiv preprint arXiv:1909.01315*, 2019. 2.3.1

[107] Xiaozhi Wang, Tianyu Gao, Zhaocheng Zhu, Zhengyan Zhang, Zhiyuan Liu, Juanzi Li, and Jian Tang. KEPLER: A unified model for knowledge embedding and pre-trained language representation. *Transactions of the Association for Computational Linguistics*, 9:176–194, 2021. doi: 10.1162/tacl_a_00360. URL https://aclanthology.org/2021.tacl-1.11. 2.1, 2.3.1, 3.1, 3.4.1, 3.2, 6.1

[108] Zhiguo Wang, Patrick Ng, Xiaofei Ma, Ramesh Nallapati, and Bing Xiang. Multi-passage BERT: A globally normalized BERT model for open-domain question answering. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5878–5882, Hong Kong, China, 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1599. URL https://aclanthology.org/D19-1599. 5.2.1

[109] Zhiguo Wang, Patrick K. L. Ng, Ramesh Nallapati, and Bing Xiang. Retrieval, re-ranking and multi-task learning for knowledge-base question answering. In *EACL*, 2021. 4.3

[110] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*, 2019. URL https://arxiv.org/abs/1910.03771. 5.3.1

[111] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Huggingface's transformers: State-of-the-art natural language processing. *ArXiv*, abs/1910.03771, 2019. 2.3.1

[112] Tianbao Xie, Chen Henry Wu, Peng Shi, Ruiqi Zhong, Torsten Scholak, Michihiro Yasunaga, Chien-Sheng Wu, Ming Zhong, Pengcheng Yin, Sida I. Wang, Victor Zhong, Bailin Wang, Chengzu Li, Connor Boyle, Ansong Ni, Ziyu Yao, Dragomir Radev, Caiming Xiong, Lingpeng Kong, Rui Zhang, Noah A. Smith, Luke Zettlemoyer, and Tao Yu. Unifiedskg: Unifying and multi-tasking structured knowledge grounding with text-to-text

language models. *arXiv preprint arXiv:2201.05966*, 2022. 4.2.4

[113] Wenhan Xiong, Xiang Lorraine Li, Srinivasan Iyer, Jingfei Du, Patrick Lewis, William Yang Wang, Yashar Mehdad, Wen-tau Yih, Sebastian Riedel, Douwe Kiela, and Barlas Oğuz. Answering complex open-domain questions with multi-hop dense retrieval. *International Conference on Learning Representations*, 2021. 4.3.2

[114] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=ryGs6iA5Km. 5.3.5

[115] Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. Embedding entities and relations for learning and inference in knowledge bases. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6575. 3.1, 3.2, 6.1, 6.1, 6.3.2

[116] Fan Yang, Zhilin Yang, and William W Cohen. Differentiable learning of logical rules for knowledge base reasoning. *Advances in neural information processing systems*, 30, 2017. 6.1

[117] Liang Yao, Chengsheng Mao, and Yuan Luo. Kg-bert: Bert for knowledge graph completion. *arXiv preprint arXiv:1909.03193*, 2019. 2.1, 3.1, 6.1

[118] Xi Ye, Semih Yavuz, Kazuma Hashimoto, Yingbo Zhou, and Caiming Xiong. Rng-kbqa: Generation augmented iterative ranking for knowledge base question answering. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2022. 1, 4.1, 4.3, 4.3

[119] Wen-tau Yih, Ming-Wei Chang, Xiaodong He, and Jianfeng Gao. Semantic parsing via staged query graph generation: Question answering with knowledge base. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1321–1331, Beijing, China, July 2015. Association for Computational Linguistics. doi: 10.3115/v1/P15-1128. URL https://aclanthology.org/P15-1128. 4.1, 4.3.1

[120] Wen-tau Yih, Matthew Richardson, Christopher Meek, Ming-Wei Chang, and Jina Suh. The value of semantic parse labeling for knowledge base question answering. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 201–206, 2016. 4.1, 4.3, 6.2

[121] Donghan Yu, Ruohong Zhang, Zhengbao Jiang, Yuexin Wu, and Yiming Yang. Graph-revised convolutional network. In *Joint European conference on machine learning and knowledge discovery in databases*, pages 378–393. Springer, 2020. 1.3

[122] Donghan Yu, Chenguang Zhu, Yuwei Fang, Wenhao Yu, Shuohang Wang, Yichong Xu, Xiang Ren, Yiming Yang, and Michael Zeng. KG-FiD: Infusing knowledge graph in fusion-in-decoder for open-domain question answering. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*,

pages 4961–4974, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-long.340. URL https://aclanthology.org/2022.acl-long.340. 1.3

[123] Donghan Yu, Chenguang Zhu, Yiming Yang, and Michael Zeng. Jaket: Joint pre-training of knowledge graph and language understanding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 11630–11638, 2022. 1.3

[124] Donghan Yu, Sheng Zhang, Patrick Ng, Henghui Zhu, Alexander Hanbo Li, Jun Wang, Yiqun Hu, William Yang Wang, Zhiguo Wang, and Bing Xiang. DecAF: Joint decoding of answers and logical forms for question answering over knowledge bases. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=XHc5zRPxqV9. 1.3, 6.1, 6.3.1

[125] Wenhao Yu, Chenguang Zhu, Yuwei Fang, Donghan Yu, Shuohang Wang, Yichong Xu, Michael Zeng, and Meng Jiang. Dict-BERT: Enhancing language model pre-training with dictionary. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 1907–1918, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.findings-acl.150. URL https://aclanthology.org/2022.findings-acl.150. 1.3

[126] 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 *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5773–5784, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-long.396. URL https://aclanthology.org/2022.acl-long.396. 4.3

[127] Ruohong Zhang, Yau-Shian Wang, Yiming Yang, Donghan Yu, Tom Vu, and Likun Lei. Long-tailed extreme multi-label text classification by the retrieval of generated pseudo label descriptions. In *Findings of the Association for Computational Linguistics: EACL 2023*, pages 1092–1106, Dubrovnik, Croatia, May 2023. Association for Computational Linguistics. URL https://aclanthology.org/2023.findings-eacl.81. 1.3

[128] Shuai Zhang, Yi Tay, Lina Yao, and Qi Liu. Quaternion knowledge graph embeddings. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 2731–2741, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/d961e9f236177d65d21100592edb0769-Abstract.html. 3.2

[129] Zhengyan Zhang, Xu Han, Zhiyuan Liu, Xin Jiang, Maosong Sun, and Qun Liu. ERNIE: Enhanced language representation with informative entities. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1441–1451, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1139. URL https://www.aclweb.org/anthology/P19-1139. 2.1, 2.3.1, 2.1

[130] Mantong Zhou, Zhouxing Shi, Minlie Huang, and Xiaoyan Zhu. Knowledge-aided open-domain question answering. *arXiv preprint arXiv:2006.05244*, 2020. URL https://arxiv.org/abs/2006.05244. 5.3.2

[131] Zhaocheng Zhu, Shizhen Xu, Jian Tang, and Meng Qu. Graphvite: A high-performance cpu-gpu hybrid system for node embedding. In *The World Wide Web Conference*, pages 2494–2504, 2019. (document), 3.2