

國立臺灣大學電機資訊學院資訊網路與多媒體研究所

碩士論文

Graduate Institute of Networking and Multimedia

College of Electrical Engineering and Computer Science

National Taiwan University

Master Thesis

應用自動驗證技術提升自然語言程式化問答系統可靠性之綜合研究

Programming Natural Language for Strengthening QA
Reliability through Automatic Validation

任恬儀

Tien-Yi Jen

指導教授：陳信希 博士

Advisor: Hsin-Hsi Chen, Ph.D.

中華民國 112 年 8 月

August, 2023



Acknowledgements

首先，我要向我尊敬的指導教授陳信希教授表達最深的感謝。感謝老師提供優質的學術環境，更感謝他在幾年間無論多忙都仍撥空指導學生提供寶貴的建議和無私的幫助，使我得以完成這篇論文。他的學術造詣和個人品格，都對我影響深遠。我也要感謝翰萱學長及安孜學姊，除了在討論中提出不同看，還願意在百忙之中播出時間，給予我研究方向和實作上的建議，他們的耐心指導和無私分享，讓我在學術道路上得以成長和進步。

此外我也要感謝實驗室的成員的提攜與陪伴，感謝博班學長們聖倫、建宏、柏君。他們在學術、生活、生涯規劃中，無私的支援和幫助，讓我感受到了學術追求以外的深厚情誼。

感謝同組的哲韋、彥斌，我們共同面對挑戰，並一起解決問題。感謝我們的網管佑恩、韋霖、家誠、淙軒，解決我在研究中製造或是遇到的各種設備問題。我也要向實驗室的同學們，表示感謝。我們一起分享快樂，共同度過難關，他們的協助與支持讓我在學術路途上更堅定。

最後，我要感謝所有對我研究進行過直接或間接參與和支持的人。無論是在學術上給予支援的教師，還是在生活上給予幫助的朋友，他們的支持使我能夠成功完成研究。



摘要

大型語言模型 (LLMs) 不僅革新了自然語言處理 (NLP) 領域，也為實際應用帶來了重大變革。儘管有這些進步，像程序生成這樣的領域仍然具有挑戰性。本論文專注於生成兩種類型的程序：數學程序和知識圖譜問答 (KGQA) 程序。

對於數學程序，我們的工作提出了一種新穎的回收數值數據擴增 (RNDA) 方法，該方法自動生成高質量的訓練實例與程序。實驗結果顯示，用擴增數據訓練的模型可以達到最先進的性能。

與此同時，在 KGQA 程序的領域，我們提出了一種反向生成的驗證方法以提高可靠性。實驗表明，這種方法也可以提高 ChatGPT 在此任務的性能。

總的來說，該研究通過引入新方法，描繪了程序生成的範式轉變，專注於改善數學和 KGQA 程序。這些發現為未來的研究提供了一個有前景的基礎，目標是充分利用大型語言模型。

關鍵字：知識庫問答、數學問題、大型語言模型



Abstract

Large Language Models (LLMs) have revolutionized not only the field of Natural Language Processing (NLP) but also brought significant changes to real-world applications. Despite these advancements, certain realms like program generation have been challenging to leverage. This thesis concentrates on the generation of two types of programs: Math programs and Knowledge Graph Question Answering (KGQA) programs.

For the math program, our work proposes a novel recycling numeracy data augmentation (RNDA) approach that automatically generates high quality training instances with programs. Experimental results show that the model trained on the augmented data could achieve the state-of-the-art performance.

Meanwhile, in the realm of KGQA programs, we propose a reverse generation-based validation to enhance reliability. Experiments show this approach can also improve the performance of the task on the ChatGPT.

In essence, the research delineates a paradigm shift in program generation through the

introduction of new methods, focusing on the betterment of Math and KGQA programs.

The findings offer a promising foundation for future research aimed at leveraging Large

Language Models to their fullest potential.

Keywords: Knowledge Graph Question Answering, Math Word Problem Solving, Large Language Model





Contents

	Page
Acknowledgements	i
摘要	ii
Abstract	iii
Contents	v
List of Figures	ix
List of Tables	x
Chapter 1 Introduction	1
1.1 Math Word Problem	2
1.2 Knowledge Graph Question Answering	4
1.2.1 Background	4
1.2.2 Motivation	6
1.3 Thesis Organization	9
Chapter 2 Related Work	10
2.1 MathWord Solving	10
2.2 Knowledge Graph Question Answering	12
2.2.1 Knowledge Graph	12
2.2.2 KGQA Dataset	12

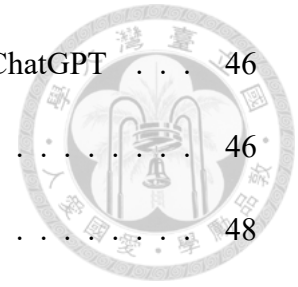


2.2.3	Relation Extraction Approach	13
2.2.4	Chinese Knowledge Graph Question Answering Dataset	13
2.3	Models	14
2.3.1	LSTM	14
2.3.2	transformer	15
2.3.3	Bert	16
2.3.4	t5	17
2.3.5	GPT3.5	17
Chapter 3	Programs	18
3.1	Programs of MathWord Problem	18
3.2	Programs of KBQA	19
Chapter 4	Methodology	20
4.1	Method of Automatic Augmentation with Validation of Math Programs	20
4.1.1	Recycling Data Augmentation	21
4.1.2	Preprocessing	22
4.1.3	Numeracy Data Augmentation	23
4.1.4	RNDA Process	23
4.1.5	Program Generation	24
4.2	Validation of Knowledge Graph Answering via Reverse-Based Method	25
4.2.1	Overall Architecture	25
4.2.2	Reverse Generate Validation Process	27
4.2.2.1	Ranker	29
4.2.3	Reverse Generator	29



4.2.4	Generator	29
4.2.5	ChatGPT No training approach	30
Chapter 5 Experiments		32
5.1	Experiment on Math Word Problem Solving	32
5.1.1	Experiment Setting	32
5.1.2	Overall Result of Math Word Problem	33
5.1.3	Result of RNDA	34
5.2	Experiment of Reverse Generation Base Validation of KBQA	35
5.2.1	Experiment Setting	35
5.2.2	Result of Reverse Generation Base Validation	36
5.2.3	Experiment Setting for ChatGPT	37
5.2.3.1	Prompt of ChatGPT Reverse Generator	37
5.2.3.2	Prompt of ChatGPT Generator	38
5.2.3.3	Corrector Prompt	39
5.3	Experiment of Generation after Ranking Approach of KGQA on Noisy CKBQA	40
5.3.1	Experiment setting	41
5.3.2	Experiment Result	41
5.3.2.1	Ranker Result	41
5.3.2.2	Ranker + Generator (Pegasus T5-base)	42
5.3.2.3	Ranker + Generator (Pegasus T5-small)	42
Chapter 6 Discussion and Analysis		44
6.1	Analysis of Math Word Problem	44
6.1.1	Analysis of Augmentation	44
6.1.2	Analysis of Parameter h	45

6.2	Case of Augmentation base Validation Combine with ChatGPT	46
6.2.1	Question Generation by ChatGPT	46
6.2.2	ChatGPT Generator (G)	48
Chapter 7	Conclusion	50
References		52





List of Figures

1.1	The question stem and the output program of a MathQA question. The answer can be derived by evaluating the program by each operation step by step.	3
1.2	Base KBQA system.	5
2.1	Sequence to sequence LSTM.	14
2.2	Transformer single encoder and decoder.	16
4.1	Recycling Numeracy data augmentation with symbolic verification.	21
4.2	The question in Figure 1.1 with preprocessing.	22
4.3	Overview of Our Model Architecture.	26
4.4	Overview of our training-free model architecture.	30
6.1	Verification on the training and test data. The horizon axis denotes the number of generated answers, while the vertical axis denotes the number of data being verified.	45



List of Tables

4.1	An example from MathQA question stem and the output program. The answer can be derived by evaluating the program step by step.	28
5.1	Overall performances of Math solving Experiments.	34
5.2	Performances of different recycle generation settings (h).	34
5.3	Statistics of Our Dataset in terms of Rating Score and Its Variance	36
5.4	Performance Evaluation of Our Modified Ranker	41
5.5	Performance Evaluation of Ranker + Pegasus T5-base	42
5.6	Performance Evaluation of Ranker + Pegasus T5-small	42
6.1	Comparison of the diversity of each dataset (Number of different types of question and different formulas.	44
6.2	The sample of different versions of questions.	47
6.3	Comparison of the reverse generator result.	47
6.4	Comparison of the ChatGPT and Ranker result under 100 instances. . . .	48



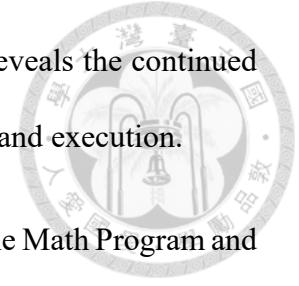
Chapter 1 Introduction

Recently, large language models have catalyzed a revolution in the field of Question Answering. Programs, the lifeblood of these models, serve as critical conduits, bridging the gap between the language model and human users. Their importance cannot be overstated, as the automatic execution of these programs can, in some instances, not only match but even outperform human capabilities, amplifying the utility and effectiveness of AI tools.

Programs are the languages that underpin functions, each containing a myriad of specific actions. They are predominantly task-oriented, meticulously designed to accomplish specific goals. The creation and implementation of these programs are vital, as they have the power to vastly expand the capacity of a language model. They not only allow these models to perform complex tasks but also adapt to new challenges and learn from their past actions. The importance of these programs becomes more apparent when we consider the limitations of AI assistants without them, which would be reduced to merely soliciting information from users.

However, even with the indispensable nature of programs, it's worth acknowledging that even the most advanced models, like ChatGPT, occasionally encounter errors, particularly with complex algebraic problems. On the other hand, the logical reasoning implied

in the programs can also help the models solve the question. This reveals the continued need for development and refinement in the field of program design and execution.



In this thesis, we will delve deeper into two types of programs: the Math Program and the Knowledge Graph Searching Program. These two examples will shed light on both the potentials and limitations inherent in current AI technology, and further underline the fundamental importance of programs in the evolving landscape of AI.

1.1 Math Word Problem

The study of math word problem solving is aimed at transforming a textual question into a numerical answer, requiring not only mathematical inference but also a high-level natural language understanding with various real-world background knowledge and several steps of operations to compute the answer. [Dua et al. \(2019\)](#) list several models and none of them can deal with symbolic reasoning. For facilitating the model to learn to solve the math word problems, [Amini et al. \(2019\)](#) introduces a symbolic language to represent the math equation in MathQA, a dataset for interpretable word problem solving. In MathQA, the intermediate answer of each question is a human-annotated *program* consisting of a series of operations. The final answer of a question, which is usually a number, can be computed from the program deterministically. Compared with the end-to-end question-answer annotation, the programs provided in the MathQA dataset offer more information for the model to learn the process of arithmetic reasoning.

For expressing the numeracy, the symbolic language of MathQA includes two kinds of symbols, operators and numbers. The operators are words for fundamental calculation such as addition, log and exponential, and words with the meaning of background

<p>Input in a school of 650 boys , 44 % of muslims , 28 % hindus , 10 % sikhs and the remaining of other communities . how many belonged to the other communities ?</p>
<p>Intermediate Program add(n1, n2) add(n3, #0) subtract(100, #1) multiply(n0, #2) divide(#3, 100)</p>
<p>Final Answer 117</p>



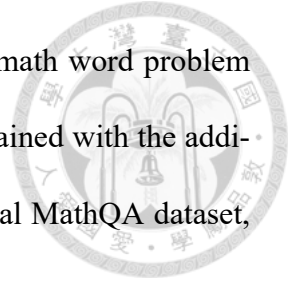
Figure 1.1: The question stem and the output program of a MathQA question. The answer can be derived by evaluating the program by each operation step by step.

knowledge such as “triangle_area” and “gcd”. The numbers can be categorized into three types, i.e., numbers given in the question, numbers generated by the former operations, and constants like π and ϵ . Figure 1.1 shows an example.

Generating the QA pairs in the MathQA style is not an easy task. Even human annotators cannot always compose a correct program for a question. In the past, the data was composed by crowd workers, and the quality of each program was further verified by the accuracy of the corresponding answer. In this way, only 29,837 training instances were created in the MathQA dataset. In contrast, AQuA (Ling et al., 2017), a much larger dataset annotated in the end-to-end style for math word problem solving, consists of about 100,000 training instances.

In this work, we aim to provide a framework for automatically generating the training instances in the MathQA style. We propose a novel recycling numeracy data augmentation (RNDA) approach and obtain additional questions with their intermediate answers (programs) and final answers. We further employ symbolic verification on the automatically augmented instances for ensuring their quality. The augmented data, along with

the original MathQA training data, can be used to train models for math word problem solving. Experimental results show a sequence-to-sequence model trained with the additional augmented data outperforms the model trained with the original MathQA dataset, confirming the effectiveness of our approach.



Sum up all the above, our contributions are threefold.

- We propose a novel recycling approach to numeracy data augmentation that automatically suggests the reasoning process for the existing datasets in math word problem solving.
- With the augmented data as supplementary training instances, our model for math word problem solving achieves the state-of-the-art performance.
- We release AQUA*, a large dataset annotated with symbolically verified programs for math word problem, as a resource for the research community.

1.2 Knowledge Graph Question Answering

1.2.1 Background

Knowledge Graph Question Answering is a special type of question answering. Same as normal question answering, an instance of knowledge graph question answering includes a question and the answer to the question. The thing different from the normal question answering is that there is a Knowledge Graph for all of the questions in or out of the Dataset.

A Knowledge Graph is a representation of information in the form of interconnected

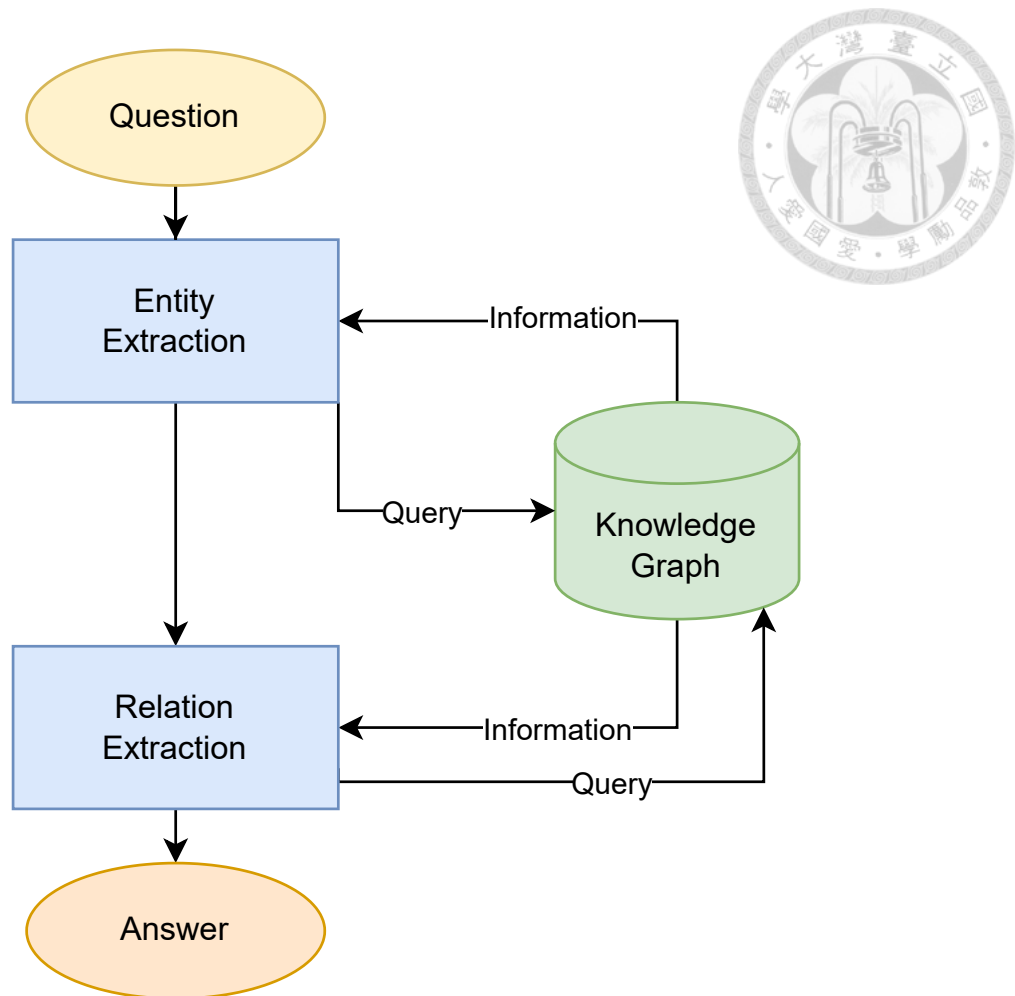
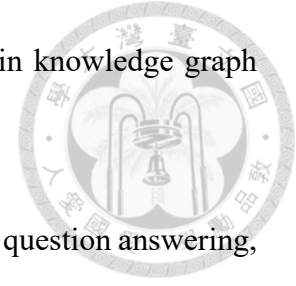


Figure 1.2: Base KBQA system.

entities and their relationships, often structured as triples. Given a Knowledge Graph KG , it will consist of many relations, $R = r_1, r_2, \dots, r_m$, entities, $E = \{e_1, e_2, \dots, e_n\}$ and triples, $T = \{t_1, t_2, \dots, t_l\}$. A triple $(t \in (e_i, r_k, e_j/l))$ consists of three elements: a subject, a predicate, and an object, where the subject and object represent two entities, and the predicate defines the relationship between them. In the advanced new type of knowledge graph, the object may sometimes be literal instead of an entity. A literal may be a number, a string, a date, and so on. It cannot be an entity, since it does not have a unique ID and may represent several different things in different triples.

Most of the information in the Knowledge Graph is irrelevant to the question of a single instance. Moreover, unlike text question answering Knowledge graph may include

millions of entities and billions of triples, especially in open-domain knowledge graph answering, which makes the task more complex.



Traditionally there are three parts in the whole knowledge graph question answering, entity extraction, relation extraction and knowledge graph execution. So a full pipeline of a KBQA process should be like Fig 1.2.

Entity extraction is a process to find several entities related to the question. A relation extraction takes the information of those entities, then finds related relations and finally gets the answer. There are two main methods to solve relation extraction task, information retrieval and semantic parsing. Information retrieval includes encoding questions and encoding the information of the related knowledge graph and then assigning a score between the information and the question to choose a set of entities or literals in the knowledge graph. Semantic parsing approach needs a middle type (e.g., program) to express the path to find the answer on the knowledge graph. The knowledge graph execution may differ from dataset to dataset. Previously the Knowledge graph execution only contains traveling through nodes by hop. However with more and more good approaches was created, simply traversing through the nodes is not enough challenging. Some of the datasets also include functions like count and argmax, in the knowledge graph execution step, which need further efforts to solve them.

1.2.2 Motivation

Recent advancements in Large Language Models (LLMs) have sparked significant developments not only in the field of Natural Language Processing (NLP) but also across a range of research disciplines within Artificial Intelligence. Despite these strides, however,

there are certain NLP tasks that remain resistant to the benefits offered by LLMs, with Knowledge Graph Question Answering (KBQA) standing as a prime example.

KBQA comprises two fundamental components: the knowledge graph (KB) and question answering. Regardless of the ever-growing volume of pretraining data for LLMs, these models cannot fully encompass the entirety of a KB. This is primarily because the creation of a triple in a KB is significantly less resource-intensive than the integration of new content into the LLM's pretraining phase.

LLMs have proven their worth in question-answering tasks, but challenges remain when it comes to merging QA and Knowledge Graphs. Historically, this task was divided into two sub-tasks: entity candidates extraction and relations extraction, as outlined in previous studies. This thesis primarily focuses on the challenges that arise in the context of relation extraction. Within this field, two primary methods have been used to address these issues: Information Retrieval and Semantic Parsing.

The Information Retrieval approach assigns a score to each candidate path, selecting the highest scoring one as the answer. This method ensures the validity of the answer and avoids situations where no answer is provided. With the emergence of sophisticated language models in recent years, however, such as T5 and BART, the Semantic Parsing approach - generally viewed as a generative task - can also harness these models.

New designs of sequence to sequence language models, which reference the Information Retrieval result, within the Semantic Parsing approach consistently outperforms Information Retrieval, even when an answer is not always provided.

This encourages us to do the experiment of such approach on another task, noisy knowledge graphs. It was characterized by duplicated information, pose unique challenges

to the field of KBQA. Not every method works under such circumstances.

Apart from that, interestingly with the advent of the T5 classification model, even the Information Retrieval approach can now take advantage of these cutting-edge language models. Research has revealed that the ranking approach outperforms the Semantic Parsing approach, with no loss of execution steps.

However, a new challenge emerges: regardless of the confidence level, a ranker always delivers an answer, which could potentially undermine the model's reliability. In certain scenarios, predicting "unknown" is more desirable than predicting an incorrect answer, as these "unknown" gaps can be addressed through human intervention. For some tasks, a simple threshold can help resolve this issue. Yet, in KBQA, the score is primarily driven by the question and cannot be easily determined by a straightforward threshold.

In this work, we propose a validation method rooted in the recursive generation between question and candidate path. While language models may not be proficient at generating flawless programs, their inherent strength in natural language understanding and generative capabilities can still be leveraged. Experimental evidence suggests that our approach can benefit from a language model's capacity to verify meaning.

Additionally, our method can seamlessly integrate with state-of-the-art large language models, such as ChatGPT, thereby enhancing the performance and efficiency of our approach. This integration is possible due to ChatGPT's superior natural language generation capability and its capacity for inline fine-tuning.

1.3 Thesis Organization



The rest of this thesis is organized as follows. Chapter 2 summarizes the related work of this study. Chapter 3 introduces the programs that were used in our experiments. In Chapter 4, we elaborate on the methodology of our extraction and generation models. Experimental results of each model are shown in Chapter 5. Further discussion and analysis of our work are presented in Chapter 6. We conclude the thesis and future work in Chapter 7.



Chapter 2 Related Work

In this chapter, we'll introduce related works about the dataset, the approach and the model we use.

2.1 MathWord Solving

The early math word problems are relatively easier. They were done by mapping (Brow, 1964) or simple operations like addition and subtraction (Bakman, 2007; Briars and Larkin, 1984; Hosseini et al., 2014). To deal with the recent math word problems (Roy et al., 2015), more types of operations and steps would be favorable.

AQuA (Ling et al., 2017) is a dataset with questions selected from the GRE and GMAT, which consists of multiple-choice numerical answers and rational textual answers. MathQA (Amini et al., 2019), a subset of AQuA, was labeled with the math symbolic language by crowd workers. In the procedure of Amini et al. (2019), to assure the quality of the annotations, constraints were added in the annotation system. However, no matter what constraints were added, the basic requirement to operate the annotation system is the human resource from annotators. Geva et al. (2020) did automatically data generation works for Drop (Dua et al., 2019) dataset. However, they did it by generating textual data for the numerical data that was generated prior to it. Apart from that, MathQA contains

more operations than it.

[Chen et al. \(2020\)](#) propose a different architecture based on the MathQA dataset. They observed that there are about 30% noisy data in MathQA. To deal with this problem, they propose two metrics, execution accuracy and operation sequence accuracy, to evaluate the accuracy of the model tested on MathQA dataset. Execution accuracy measures the correctness of the final multi-choice answer after running the execution, and operation sequence accuracy measures the ratio of the generated symbolic sequences that match the ground truth symbolic sequence exactly. They showed that the models optimized toward sequence accuracy tend to achieve a higher accuracy because the noisy test data and such bias does not exist in the execution accuracy. They also found that some of the symbolic answers in MathQA are wrong.

In this thesis, to reduce the influence of the noisy data, we use execution accuracy for the evaluation. Since [Chen et al. \(2019\)](#) uses more lenient version of operation accuracy, their results are not suitable to compare with ours directly. That is, only the final value of the generated sequence must match the ground truth symbolic sequence.

With the creation of the datasets such as MathQA and AQuA, more and more research works on deep learning and sequence to sequence model ([Huang et al., 2018a,b](#); [Ling et al., 2017](#)). That encourages us to enlarge the dataset. In this thesis, we will expand the MathQA-type data to AQuA Dataset with recycle method. In particular, the intermediate symbolic forms offered in MathQA show the effectiveness of models to learn the ability to math word problem solving. How to obtain more question and program pairs without the cost of human annotation raises a new challenging issue. For this reason, this work proposes a novel approach to numeracy data augmentation.

2.2 Knowledge Graph Question Answering



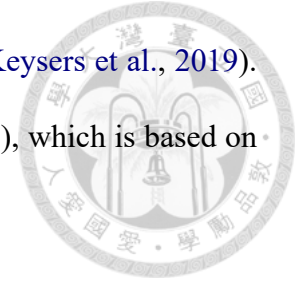
2.2.1 Knowledge Graph

Knowledge graph question answering (KGQA) primarily leverages three principal knowledge graphs, namely Freebase([Bollacker et al., 2008](#)), WikiData([Vrandečić and Krötzsch, 2014](#)), and DBpedia([Auer et al., 2007](#)). Most question answering (QA) datasets are constructed on one of these platforms. However, Freebase became temporarily inaccessible due to its closure in 2016. While this presented a setback, it also created an opportunity. As WikiData and DBpedia are still available online, they are advantageous for large language models such as ChatGPT. This accessibility can influence experimental outcomes on QA datasets derived from these platforms. Therefore, we have chosen to utilize a dataset constructed on the Freebase platform.

2.2.2 KGQA Dataset

Traditionally, question answering was limited to simple queries([Bordes et al., 2015](#)) that consisted of a single triple from the knowledge graph or was conducted on a small scale([Berant et al., 2013](#)). Another constraint was the absence of annotated programs that could guide towards the answer within the knowledge graph. With increasing complexity, datasets encompassing multi-hop queries with program annotations were introduced([Dubey et al., 2019](#); [Trivedi et al., 2017](#); [Yih et al., 2016](#)). However, they remained limited to two-hop questions. The first three-hop dataset, MetaQA, was introduced by ([Zhang et al., 2018](#)). However, its limited relations were nearly solved by 2021([Atzeni et al., 2021a](#); [Ravishankar et al., 2021](#); [Thai et al., 2022](#)). Subsequently, a diverse range of

datasets has emerged since 2019(Cao et al., 2022; Gu et al., 2021; Keysers et al., 2019). For our study, we have selected the GrailQA dataset (Gu et al., 2021), which is based on Freebase and comprises more comprehensive questions.



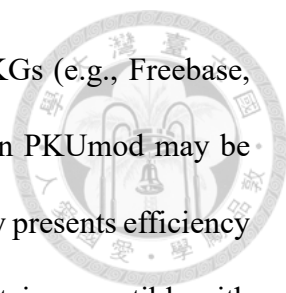
2.2.3 Relation Extraction Approach

As outlined in the introduction (Section 1.2), relation extraction primarily follows two methodologies. In Information Retrieval, all candidate answers are sorted by the candidate's type, path, and words. This was initially executed by embedding learning models such as CNN or LSTM(Bordes et al., 2014; Dong et al., 2015). Numerous works have replicated this step or composition for multi-hop questions(Jain, 2016). Some recent studies have also combined modern models like Bert or GNN(Saxena et al., 2020; Wang et al., 2020). In previous years, the semantic parsing approach relied mostly on dependency and schema parsing(Cai and Yates, 2013). Now, it is executed by sequence generating models (Lan and Jiang, 2020; Sun et al., 2020), also known as generators. Increasingly, researchers are integrating both approaches to enhance performance, either by ranking after generating(Atzeni et al., 2021b) or generating after ranking(Shu et al., 2022; Ye et al., 2022). In this study, we utilize a strategy that generates after ranking.

2.2.4 Chinese Knowledge Graph Question Answering Dataset

The most prominent Chinese knowledge graph is Pkumod, developed by the Data Management Lab at the Wangxuan Institute of Computer Technology, Peking University. Since 2018, the China Conference on Knowledge Graph and Semantic Computing has based on a small dataset for its competition, referred to as CCKS. However, Pkumod does

not assign an ID for each entity, causing different entities in other KGs (e.g., Freebase, WikiData) to be viewed as the same. Consequently, some entities in PKUmod may be associated with an excess of ten thousand relations. This issue not only presents efficiency concerns but also renders some models, which require relations as input, incompatible with the dataset.



2.3 Models

In this section, we introduce various models employed in our research, each with its distinct properties and strengths in processing and understanding sequence data.

2.3.1 LSTM

Long Short-Term Memory (LSTM) networks, introduced by Hochreiter and Schmidhuber (Hochreiter and Schmidhuber, 1997), is a type of Recurrent Neural Network (RNN) specifically designed to learn and remember long sequences of data (Sutskever et al., 2014). Due to their ability to handle time series or sequential data, LSTM networks have been widely utilized in sequence-to-sequence models. Figure 2.1 illustrates the structure of a sequence-to-sequence LSTM model.

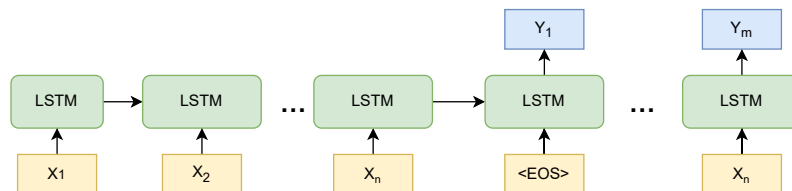


Figure 2.1: Sequence to sequence LSTM.

The core functions in an LSTM unit are as follows::



Forget Gate:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

Input Gate:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

Core status:

$$\hat{C}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c)$$

$$C_t = f_t \odot C_{t-1} + i_t \odot \hat{C}_t$$

Output Gate:

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

2.3.2 transformer

The Transformer model, first introduced in the paper “Attention is All You Need” (Vaswani et al., 2017), simplifies architecture by solely relying on attention mechanisms rather than more complex structures such as RNNs or CNNs. A Transformer typically consists of several layers of encoders and decoders. The structure of a Transformer with a single encoder and decoder is depicted in Figure 2.2.

In contrast to RNNs, Transformers offer superior performance in parallel training as they do not rely on previous outputs during training. Moreover, due to their direct attention mechanism, Transformers exhibit better performance with longer texts compared to RNNs.

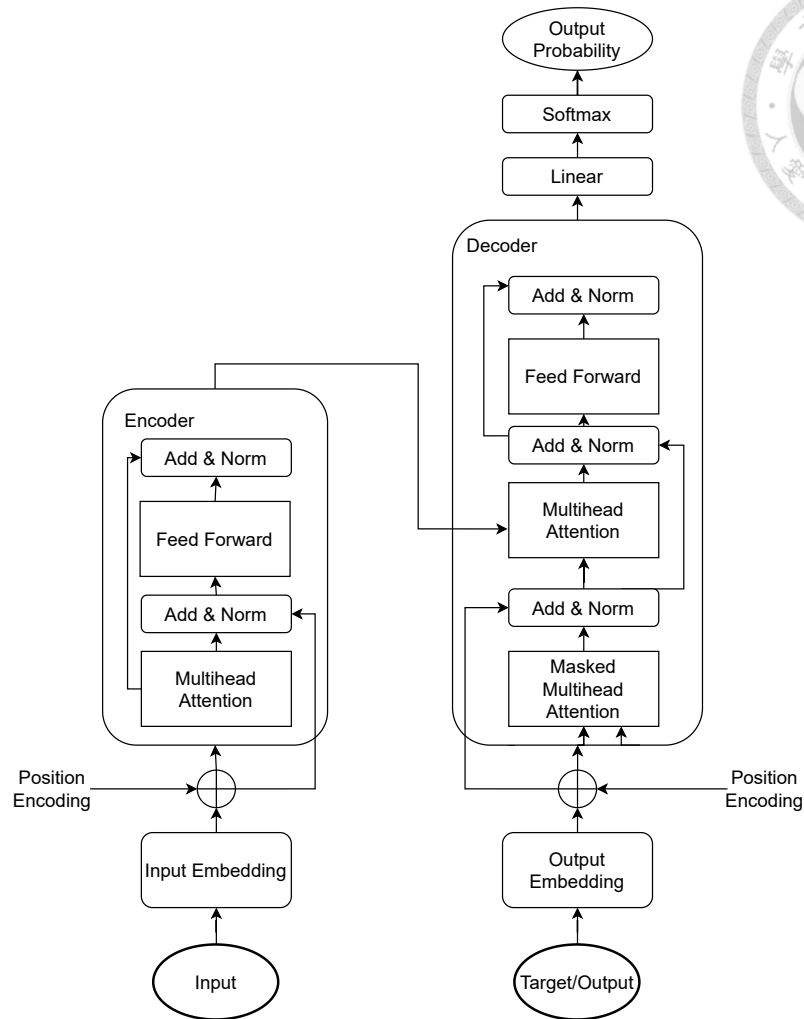


Figure 2.2: Transformer single encoder and decoder.

2.3.3 Bert

The Bidirectional Encoder Representations from Transformers (BERT) was introduced by Devlin et al. (Devlin et al., 2018). BERT's architecture is based on Transformer encoders. The model was trained on large datasets using the Masked Language Model (MLM) and Next Sentence Prediction (NSP) strategies. By introducing a novel approach to language understanding, BERT marked a milestone in the field of Natural Language Processing (NLP).

2.3.4 t5

The Text-to-Text Transfer Transformer (T5) was proposed by Raffel et al. (Raffel et al., 2020). It is a Transformer-based model that adopts a unified text-to-text approach, where every NLP task is treated as a text generation problem. This innovative perspective allowed the model to deliver high performance across diverse tasks, as it could transfer knowledge learned from one task to another more seamlessly.

2.3.5 GPT3.5

The Generative Pretrained Transformer 3.5, also known as ChatGPT, was designed to handle tasks given by specific instructions (OpenAI, 2021). Like other models, ChatGPT also underwent pretraining on large datasets. Additionally, it used reinforcement learning, accompanied by an evaluation model, to enhance its ability to execute instructions more accurately. Despite the breakthroughs it brought about in NLP tasks, ChatGPT still faces challenges in logical reasoning and answering questions that require complex understanding.





Chapter 3 Programs

3.1 Programs of MathWord Problem

This thesis introduces mathematical programs, as delineated by (Amini et al., 2019).

Each program comprises several operations, consisting of three types of words:

Operator: Every operation includes an operator. There are a total of 79 different operators, subdivided into:

- **Simple operators:** These include basic arithmetic operations such as addition, subtraction, and multiplication.
- **Complex math operators:** These involve more intricate operations like choice and square root.
- **Background knowledge operators:** These incorporate domain-specific operations like `triangle_area_three_edges`, `volume_cone`, and `union_prob`.

Constant: Constants are words specifying a numerical value, for instance, `min_to_hour`.

Variable: Variables represent numeric words in the question and are typically represented as n_1, n_2, \dots, n_k .

3.2 Programs of KBQA



Previous datasets utilized SPARQL (Harris and Seaborne, 2013), a distinct SQL dialect for executing queries on knowledge graphs, as the program for KBQA questions. However, a program for KBQA does not necessarily need redundancy words for SQL. To this end, Gu et al. (2021) introduced a new program for KBQA, referred to as s-expression. This program enhances readability and compositionality. The s-expression for KBQA consists of relations, entities, literals, and operators. The operators in the s-expression for KBQA are as follows:

AND: Used to unionize two sets of nodes (i.e., entity/literal).

JOIN: Utilized to add a relation onto a set of nodes.

ARGMAX/ARGMIN: Extracts the maximum/minimum node from a set of nodes.

LT(LE/GT/GE): Returns a Boolean value determining whether the node is $<$ (\leq / $>$ / \geq).

COUNT: Provides an integer indicating the number of elements in the set.



Chapter 4 Methodology

In this chapter, we present our methodology for augmenting math programs using a validation approach. We detail various configurations of our reverse validation method for knowledge graph question answering, including both training and non-training (in-line fine-tuning) approaches. The methodology for the math programs is described in Section 4.1, while the validation method for knowledge graph question answering and the non-training approach are outlined in Sections 4.2 and 4.2.5 respectively.

4.1 Method of Automatic Augmentation with Validation of Math Programs

In order to obtain more training pairs in the MathQA style, we propose a novel RNDA approach. Our idea is to automatically compose the program for each training instance that is already annotated in the end-to-end style. With the program, a large amount of the end-to-end instances can be created from math problem datasets such as AQuA.

For the task with a lot of datasets unlabeled with the program like math word problems, this augmented method can improve the problem of small dataset. The details of our methodology are described as follows.



4.1.1 Recycling Data Augmentation

To ensure the correctness of the machine-generated programs, we adopt the large-scale end-to-end dataset, AQuA, as the material. The recycle generation has been shown its effectiveness in natural language generation (Hinson et al., 2020). We first train M^1 , a model for math word problem solving, on MathQA and employ M^1 to compose a number of candidate programs for each question in AQuA, where every question is labeled with the final answer. Then we select all correct programs by verifying the value derived from every candidate symbolic program, resulting, AQuA², a new set of instances with annotation in the MathQA style. Figure 4.1 illustrates the procedure.

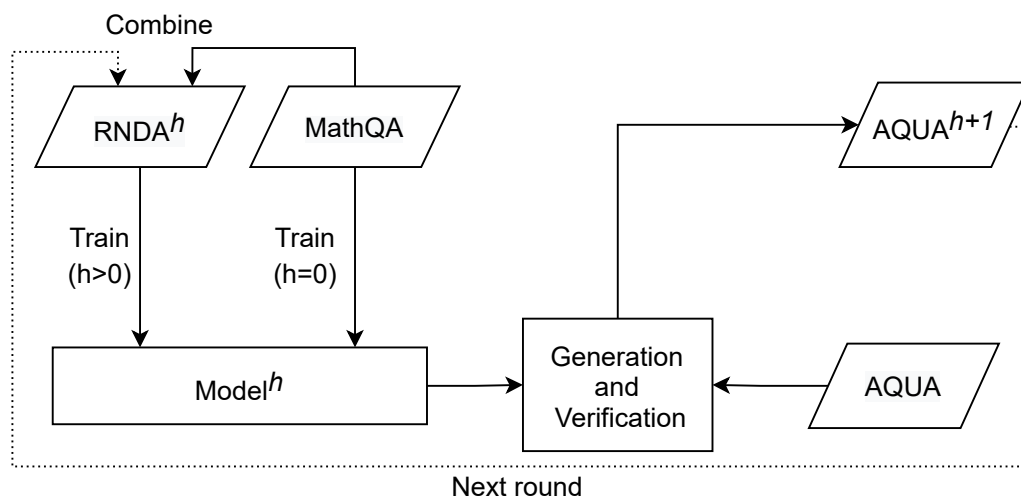


Figure 4.1: Recycling Numeracy data augmentation with symbolic verification.

With the genuine data from MathQA and the semi-pseudo labeled data from AQuA², we can train a new model M^2 , and further apply the same procedure to obtain AQuA³ and M^3 , AQuA⁴ and M^4 , ..., AQuA^h and M^h , iteratively. We iteratively build models and generate new data based on the current best dataset, and get a new one. Our recycling data augmentation approach differs from traditional self-training in an important aspect. In self-training, the correctness of the pseudo-labels predicted by the model is unknown. The

in a school of n_0 boys , n_1 % of muslims , n_2 % hindus , n_3 % sikhs and the remaining of other communities . how many belonged to the other communities ?



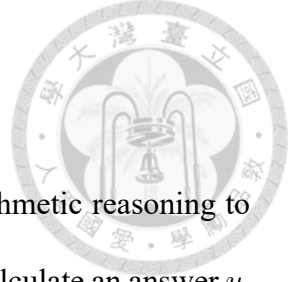
Figure 4.2: The question in Figure 1.1 with preprocessing.

information applied to control the quality of the new data, e.g., prediction confidence, is also made by the model and inherent in circular reasoning. In contrast, the quality of the augmented data in our approach is symbolically verified with independent and genuine information. Formally, given a candidate program, we evaluate it and obtain the final answer (a numerical number) \hat{y} , and compare \hat{y} with the golden answer given in AQuA. The program is regarded as correct if the final answer it derives equals to the golden one.

4.1.2 Preprocessing

In the sequence to sequence models, words split by whitespaces are first encoded into their binary indices in the dictionary before being inputted to any other layers. Even a character of difference leads to a different index in the dictionary. The model views each index of a word as a completely different thing. In order to deal with this problem, while preprocessing, we replace the numerical words (e.g. 1, 41, 321124) with indexes such as n_1 , n_3 , and n_{10} . Unlike [Luong et al. \(2015\)](#), which aligns rare words to words in the source text, our addressing reduces the complexity of different numerical words and encodes the positional information of the numerical words. Figure 4.2 shows an example.

<https://www.overleaf.com/project/64514262a61a4382fb0e703c>



4.1.3 Numeracy Data Augmentation

Formally, we define a program that is used to represent the arithmetic reasoning to answer a question in the MathQA dataset as Θ , which can derive to calculate an answer y . This answer can be checked with a ground truth \hat{y} simply and automatically by computing. For each question, we can generate multiple math representations Θ along with their probabilities, and calculate all possible y values.

In this work, we aim to automatically generate more questions with their golden calculation answers \hat{y} and programs that derive to the golden answers. We use AQuA as the source for data augmentation since MathQA is rewritten from the subset of AQuA. In addition, AQuA is a pure math problem dataset with numerical options and answers, so it is one of the most suitable datasets for the first step of expanding.

4.1.4 RNDA Process

We iteratively use the generative data, $AQuA^h$ to build a new model M^h and generate $AQuA^{h+1}$. Some annotations lead to the right answers but we can obviously observe that they include redundant operations, such as dividing 0 by a number or adding 0 to a number. In addition, some of the operations will not be used in the later operations or the answer. These two types of redundant operations should not be in our data. As a result, in the generation step, we remove them and keep the other useful operations.

In this way, we generate $AQuA^2$ by the above process and train a new model M^2 with the combination of the original MathQA and $AQuA^2$. This procedure can be applied iteratively to generate $AQuA^3$ and M^3 , ..., $AQuA^h$ and M^h . The best value of h is a

hyper-parameter to explore.



4.1.5 Program Generation

The model for program generation is a sequence to sequence one based on Transformer with an attention layer for decoding. Let $\Theta = \phi_1(\theta_1)\phi_2(\theta_2)\dots\phi_N(\theta_N)$ be a whole annotation of a math problem. An annotation contains N operations. Each operation consists of one operator (e.g., add and sqrt) and several numerical words (e.g., n1 and constant_1). ϕ_i is the operator of the operation i , θ_i is the numerical words of the operation i , and $\Theta_T = \phi_1(\theta_1)\phi_2(\theta_2)\dots\phi_T(\theta_T)$ means the operations from 1 to T .

The ranking score ψ of the operator ϕ_t and numerical words θ_t for the encoder output σ and previous order words Θ_{t-1} is:

$$\psi_{\sigma, \Theta_{T-1}}(\phi_T \theta_T) = D(\sigma, \Theta_{T-1}, \phi_T) \prod_{t=1}^n D(\sigma, \Theta_{T-1} \phi_T \theta_{T,1} \dots \theta_{T,t-1}, \theta_{T,t})$$

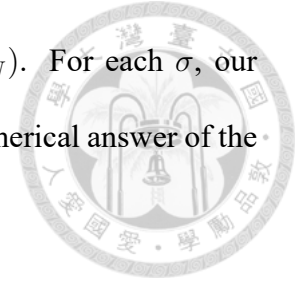
where \underline{D} means the decoder of the model, $\theta_{T,j}$ is the numerical word with index \underline{j} in θ_T and \underline{t} is the amount of the numerical words in θ_T . The ranking score Ψ of the annotation Θ is defined as follows, when encoder output is σ :

$$\Psi_{\sigma}(\Theta) = \prod_{T=1}^N \psi_{\sigma, \Theta_{T-1}}(\phi_T \theta_T)$$

The calculation value \hat{y}_T of Θ_T is :

$$\hat{y}_T = \Gamma(\Theta_T) = \phi_T(\theta_T, \gamma_T)$$

where $\gamma_{T-1} = (\Gamma(\Theta_1), \Gamma(\Theta_2), \dots, \Gamma(\Theta_{T-1}))$, and $\hat{y} = \Gamma(\Theta) = \Gamma(\Theta_N)$. For each σ , our target is to maximize $\Psi_\sigma(\Theta)$ so that $\hat{y} = y$, where y is the exactly numerical answer of the question of σ .



We also implement a parameter k , which constrains the max amount of Θ^* that can be verified. If multiple Θ^* are verified by the system, we choose the one with the highest $\Gamma(\Theta^*)$ of all of the Θ .

First, we use the MathQA model to generate ($k = 167$) outputs (Θ) for each question in the training dataset of AQuA. Then, we get the correct numerical answer (y) from the options and compare it with \hat{y} of each Θ . If one of the answers is correct, then we can give the question a math representation language sequence.

The computational complexity of our generation progress is controlled by the parameter k and the length of the program N .

$$O(\text{generation program}) = k * N$$

There's no additional computational time complexity when h increases.

4.2 Validation of Knowledge Graph Answering via Reverse-Based Method

4.2.1 Overall Architecture

Figure 4.3 demonstrates the overarching architecture of our approach.

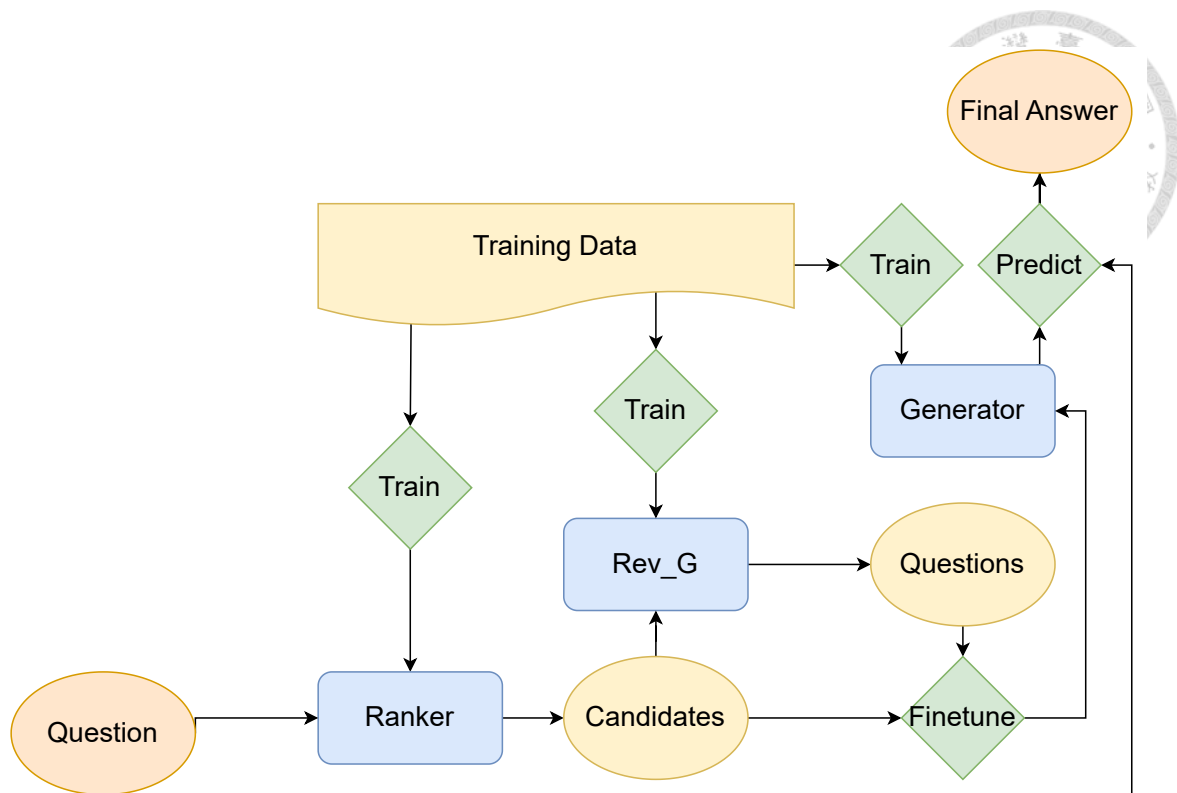


Figure 4.3: Overview of Our Model Architecture.

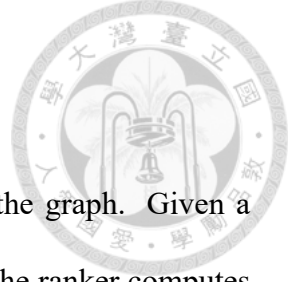
Our architecture is composed of three main components: a ranker R , a generator G , and a reverse generator Rev_G . The input to the secondary finetuning phase of generator G consists of the original question, the output from ranker R , and the output from reverse generator Rev_G .

The process for Generator G involves two stages of finetuning.

In the first stage, the finetuning is performed using the training data, while the second stage employs auto-generated data Rev_G for each instance.

This auto-generated data is produced by the ranker and the reverse generator.

This approach can combine the use of the meaning words in the program and the ability of natural language generation of the language model. It helps the language model to realize the schema of the program with natural language meaning better.



4.2.2 Reverse Generate Validation Process

The Ranker R is first trained to assign scores to the paths in the graph. Given a question Q and a set of possible paths in the knowledge graph KB , the ranker computes the top-ranked candidates C :

$$C = R(Q, KB)$$

Then, the Generator G is trained to produce programs S from a question q :

$$S = G(q)$$

Moreover, the Reverse Generator Rev_G is trained to generate questions Q from a program s :

$$Q = Rev_G(s)$$

where Q consists of multiple questions:

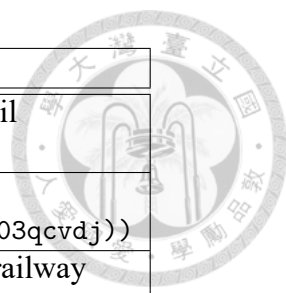
$$Q = \{q_1, q_2, \dots, q_n\}$$

Given a new question q_1 , the Ranker R is first employed to generate candidates C :

$$C = R(q, KB) \tag{1}$$

$$C = \{c_1, c_2, \dots, c_m\} \tag{2}$$

where c_i is an individual candidate path.



Data Type	Example
Question	semaphore railway line is on the rail network named what?
Candidate Program	(AND rail.rail_network (JOIN rail.rail_network.railways m.03qcvdj))
Augmented Question	what is the rail network that has a railway line called semaphore railway line?
Final Program	(AND rail.rail_network (JOIN rail.rail_network.railways m.03qcvdj))
Final Answer	["m.04rydm"]

Table 4.1: An example from MathQA question stem and the output program. The answer can be derived by evaluating the program step by step.

Next, Rev_G is used to generate the questions for each candidate:

$$Q_i = Rev_G(c_i) \quad (3)$$

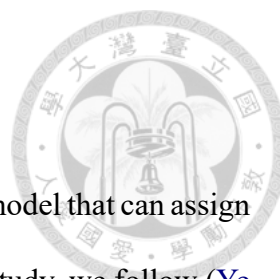
$$Q_i = \{q_{i1}, \dots, q_{in}\} \quad (4)$$

The pairs (q_{ij}, c_i) are then used to further fine-tune G .

Finally, the ultimate program is generated by:

$$S^* = G(q)$$

The reliability of the output is verified with the final program. If the output program is non-executable, it is considered as the model's decision that no suitable answer exists within the given programs. An executable program indicates the program can yield some answer on the knowledge graph. In practice, we generate k programs for validation and choose the first executable program as the final validated answer. If none of the programs are executable, it is defined as none of the programs being valid.



4.2.2.1 Ranker

A ranker could be a Graph neural network, a transformer, or any model that can assign a score between a knowledge graph program and a question. In this study, we follow (Ye et al., 2022) and use a Bert classifier as the ranker in the experiment. The Bert classifier consists of a Bert and a cross encoder of the question q and the program p to be scored:

$$score(q, p) = Linear(Bert(Cat(q, p)))$$

4.2.3 Reverse Generator

The role of the Reverse Generator is to convert a candidate program back to questions, as shown in the overall architecture (3). Depending on the generation setting, the Reverse Generator will generate several questions for each candidate. In the training approach, our Reverse Generator uses the T5 model as a base. We finetune the T5 model using the gold programs and questions in the training dataset to perform the task.

4.2.4 Generator

The Generator's role is to convert a question into candidate programs. The architecture of our Generator is also a T5 model. A Generator requires two steps of finetuning before the final generation. In the first step, the Generator is trained using the training dataset, where it is tasked with directly generating the golden program from the question. In the second step, with the candidate programs generated by the Ranker and augmented questions generated by the Reverse Generator, the Generator is further fine-tuned. See Table 4.1, for the example of a candidate program and an augmented question.) According

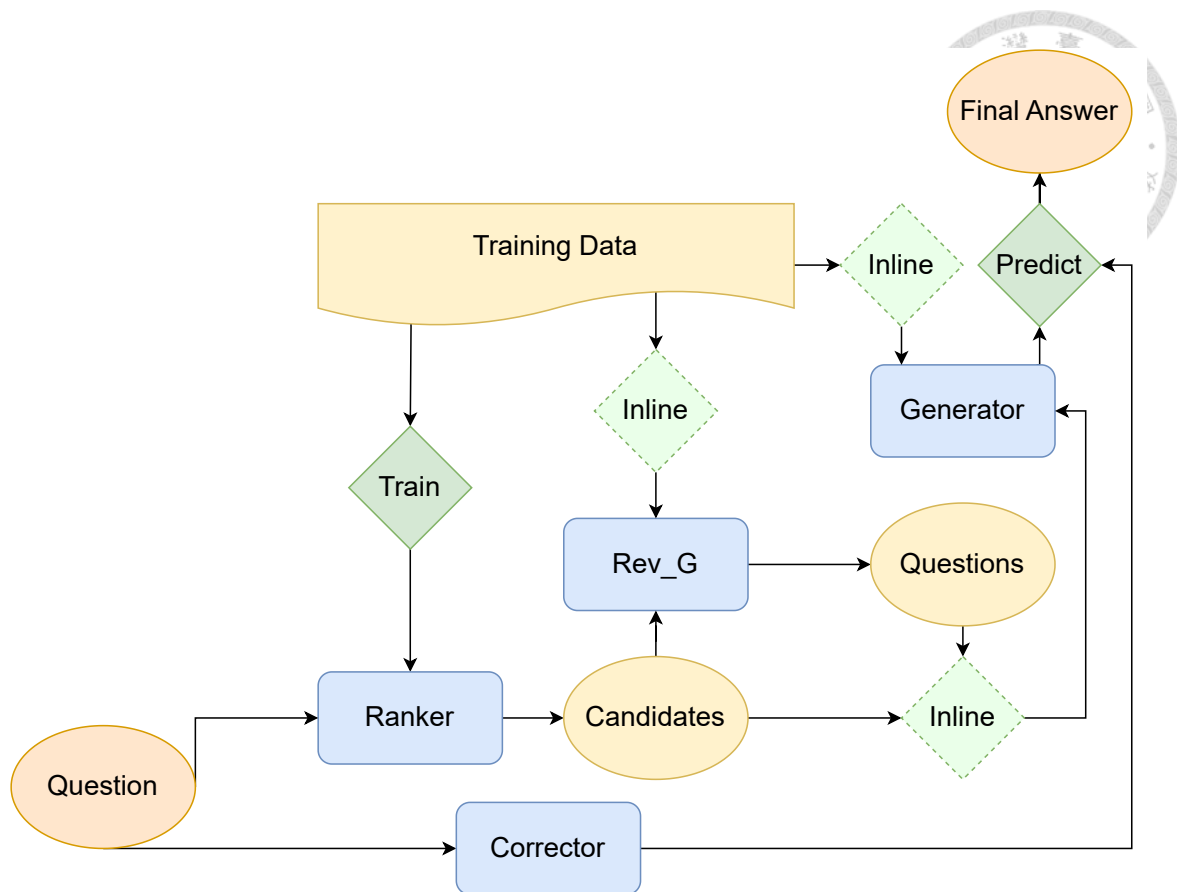


Figure 4.4: Overview of our training-free model architecture.

to the generation strategy, it generates different amounts of programs for a question.

4.2.5 ChatGPT No training approach

Departing from the T5 model, we further combine our validation process with the LLM, significantly improving the efficiency of our process. The central idea is to use inline fine-tuning to replace traditional fine-tuning on the model, leading to a training-free approach for the validation process in any Information Retrieval based approach.

In this approach, we denote ChatGPT as L . Similar to previous steps, we first use the Ranker to get candidate programs C , then use ChatGPT reverse generator L_R to generate

questions for each candidate program:

$$q'_i = L_R(c_i)$$



With the augmented question and program pairs, we use ChatGPT generator L_G to generate the final answer:

$$S^* = L_G(\{(q, c) \mid q \in Q, c \in C\})$$

This allows us to achieve a no-training version of our validation process. In addition to the components in the T5 approach, we introduce a new model in this approach, Corrector L_c . The Corrector is designed to make the input question more conversational, reflecting natural human speech:

$$L_c(q) = q'$$

Here, q' is a question with the same meaning as q . By comparing dataset questions and questions generated by the ChatGPT question generator, we found that the questions generated by ChatGPT are more conversational. Therefore, we use ChatGPT to implement the Corrector. More discussion on this topic can be found in section 6.2.



Chapter 5 Experiments

In this chapter, we introduce our experiment on automatic program generation for math word problems, specifically focusing on the reverse base method. Additionally, we will describe the subsequent experiment involving a ranking strategy applied to a noisy Chinese dataset for KGQA tasks.

5.1 Experiment on Math Word Problem Solving

5.1.1 Experiment Setting

We implement the model with the fairseq framework.¹ For the LSTM model, there are two layers for both the encoder and the decoder. The embedding dimensions of both the encoder and the decoder are 100 with dropout rates of 0.2. The dropout rates of the rest layers are set to 0.3.

For the transformer model, there are 6 attention heads and 6 layers for both encoder and decoder. The embedding dimension for encoder is 600 and for decoder is 300. The learning rate is 10^{-4} . The optimizer is Adam. The criterion is cross entropy. For the generation, the beam size is 167, the size of the full vocabulary of the math program

¹<https://fairseq.readthedocs.io/en/latest/>

language.



5.1.2 Overall Result of Math Word Problem

We compare our models with the state-of-the-art model (Amini et al., 2019; Chen et al., 2020), as shown in the first row.

In MathQA, Chapter 2, we've mentioned that some of the representations' calculation are incorrect. Thus, we can prune the instances that are not validated the correctness (includes the question with "None of above" as the correct answer) and obtain a pruned MathQA dataset, consisting of 22,181 valid training instances and we'll then call it MathQA*. The dataset made by our NDA algorithm MathQA augmentation includes 43,854 AQuA data with MathQA type annotation. We'll then call it as $RNDA_h$ which h is the round of the Augmentation process.

For testing the effectiveness of our RNDA approach, we train our model for math word problem solving with several versions of training data, the original MathQA dataset, the MathQA*, and the M_h s, and evaluate the models on the testset of MathQA. Since the performance of the models built from MathQA are better, we have the new experiment.

Experimental results show the seq2seq solver trained on the original MathQA is slightly inferior to the state-of-the-art model, and the model trained on the cleaner but smaller MathQA* is even worse. However, the model with the aid of our NDA approach achieves an Accuracy of 61.7%, making a significant improvement.



Approach	Architecture	Accuracy
Amini et al. (2019)		54.20 %
Chen et al. (2020)		55.95 %
MathQA	LSTM	53.10 %
MathQA	Transformer	56.00 %
MathQA*	LSTM	50.1 %
MathQA*	Transformer	54.3 %
RNDA	LSTM	61.7 %
RNDA	transformer	63.6 %
Ours Best	transformer	67.28 %

Table 5.1: Overall performances of Math solving Experiments.

5.1.3 Result of RNDA

To dive into the details of the results of our RNDA process, we explore different values of the hyperparameter h , which is the number of recycle generations. As shown in Table 5.2, the performance increases as the increase of h for $h < 4$. In particular, the use of only one generation of RNDA drastically improves the Accuracy from 56.00% to 63.60%, showing the benefit of additional training instances with the symbolic intermediate forms.

The number of augmented instances is decreased at $h = 4$ because more pseudo-label instances do not pass the verification, and the performance also downgrades.

h	Augmented Instances	Accuracy
0	29,837	56.00%
1	43,854	63.91%
2	56,367	66.45%
3	62,531	67.28%
4	61,842	66.89%

Table 5.2: Performances of different recycle generation settings (h).

5.2 Experiment of Reverse Generation Base Validation of KBQA



5.2.1 Experiment Setting

In order to do the ChatGPT version experiment, we selected the FreeBase dataset as our Knowledge Graph. This is because both Wikidata(Vrandečić and Krötzsch, 2014) and DBpedia(Auer et al., 2007) continue to be accessible over the internet, allowing ChatGPT to directly retrieve data from Wikidata without our intermediate processing.

In this setup, for the ranker, we adhered to the configuration described by (Ye et al., 2022) to generate to top 10 candidates. The top 10 candidates were used to generate questions, and 10 unique questions were produced for each candidate in order to do fine-tune by instance. As per (Ye et al., 2022), we ensured that the programs used were linearized.

We leveraged several versions of the T5 model for the generator, specifically t5-large(Raffel et al., 2020), t5-small, and t5-ssm-large, within an open book setting. Our experiments involving these T5 models followed three different methodologies: fine-tuning on training data alone, fine-tuning solely on instance-specific augmentation data, and a combination of both training data and instance-specific augmentation data for fine-tuning.

We conducted an additional experiment focusing on instance-specific augmentation data using t5-large, allowing us to draw comparisons with our non-training approach in the ChatGPT experiment.

In the context of T5 fine-tuning settings, we used a batch size of 32 and a beam size of 10. Further, we established an upper limit of 10 for the maximum sequence length subject

to validation.



5.2.2 Result of Reverse Generation Base Validation

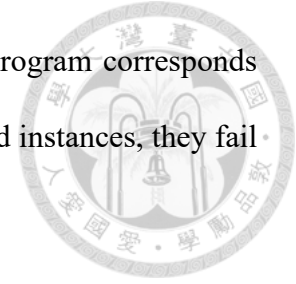
Approach	Pretrain Model	F1(All)	F1 (Exec)	Exec Ratio
Finetuned on Train only	T5-small-ssm	0.2512	0.8400	0.2990
Finetuned on Train + Augumentation	T5-small-ssm	0.6215	0.8002	0.7766
Finetuned on Augumentation only	T5-large	0.5744	0.6962	0.8251
Finetuned on Train only	T5-large	0.3061	0.8697	0.3519
Finetuned on Train + Augumentation	T5-large	0.6586	0.7873	0.8366
Finetuned on Augumentation only	T5-large-ssm	0.5825	0.6787	0.8583
Finetuned on Train only	T5-large-ssm	0.3062	0.8684	0.3526
Finetuned on Train + Augumentation	T5-large-ssm	0.6674	0.8032	0.8309
Ranker		0.6842	0.6842	1

Table 5.3: Statistics of Our Dataset in terms of Rating Score and Its Variance

We used three metrics to evaluate the performance of each approach. F1(All) is the mean F1 score of the approach on the complete dataset. In cases where all the programs cannot be executed, the F1 of that data is evaluated as 0 in F1(All), signifying that none of the data is validated by the model. F1(Exec) is the mean F1 score of the approach, disregarding the data that cannot be validated. This score indicates the performance of the instances that the model considers valid, thereby demonstrating the reliability of the approach. The Exec Ratio represents the percentage of instances that can be executed, meaning the language model identifies some of the programs as valid. This metric can be directly calculated using F1(All) and F1(Exec).

When compared to the ranker’s results, our approach significantly enhances the reliability of the model while only slightly compromising overall performance. The T5-small experiment demonstrates that our approach is also beneficial in settings with smaller model sizes. The primary issue is its inability to maintain the executability of the generated programs, meaning it tends to deem programs unreliable. The experiment finetuned on the

training set only reveals that language models can determine if a program corresponds correctly to a question. However, without the reference of augmented instances, they fail to generate suitable programs.



5.2.3 Experiment Setting for ChatGPT

In our experiment, we employed OpenAI's "gpt-3.5-turbo" as the Large Language Model (LLM). We randomly selected 100 instances for this analysis.

Given the maximum token limit of 4,000, we generated only one candidate question for each candidate, and subsequently, one final program per instance. In the context of in-line fine-tuning, we used 20 instances for the reverse generator and 3 instances for the generator due to the token limit.

5.2.3.1 Prompt of ChatGPT Reverse Generator

In the ChatGPT version of the reverse generator, we utilized a system prompt to define the task and provide samples. This approach remains consistent across different programs.

Here's the task explanation prompt:

You are a translator which translates an expression of specific logical form to a natural language question. To smooth out the question, you may ignore some of the words to avoid duplicate word. Here's the description of the functions in the logical form:

Refer to Section 3.2 for the functions' description. The system prompt will be con-

tinued with the samples. The following is a sample instance:



Sample Input : (AND measurement_unit.measurement_system (JOIN measurement_unit.measurement_system.current_density_units ampere_per_square_metre))

Sample Output : what is the measurement system that uses ampere per square metre as a unit for current density?

The sample input and output are the linearized gold program and the question of an instance in the training dataset. The user prompt is in the same format as Sample Input.

5.2.3.2 Prompt of ChatGPT Generator

In the ChatGPT version of the generator, we also use a system prompt to explain the task and give a sample.

Here's the task explanation prompt:

You are a translator who translates a natural language question into an expression of a specific logical form. Here's the description of the functions in the logical form:

...

For each question, you will get several pairs of related questions and their logical form answer. Knowing that all of the logical forms in the reference are valid.

The description of the functions is also in Section 3.2. Then is the sample instances part. The following is a sample of the system prompt:



Sample Input:

Question 1: what is the name of the lighthouse that has a focal height of light of 62.0?

Logical form 1: (AND architecture , lighthouse (JOIN architecture , lighthouse , focal height of light 62.0))

Question 2: what is the name of the lighthouse that has an intensity of 62.0?

Logical form 2: (AND architecture , lighthouse (JOIN architecture , lighthouse , intensity 62.0))

...

Question 10: Which TV series season has a season number of 62.0?

Logical form 10: (AND tv , tv series season (JOIN tv , tv series season , season number 62.0))

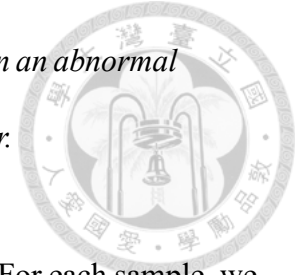
Reference the question and logical form pairs above, answer the question with logical form: Which lighthouses have lights that are 62.0 meters high?

For each k , *Logical form k* is a candidate program chosen by the ranker and *Question k* is the result of the reverse generator of Logical form k . The user prompt follows the same format as the Sample Input.

5.2.3.3 Corrector Prompt

There is no need for a program description in the corrector prompt. Consequently, the task explanation is:

You are a question corrector. The user will give you a question in an abnormal speaking order. You should return the question in normal order.



The system prompt contains several sample inputs and outputs. For each sample, we use the question and the reverse generator's output of the gold program from an instance in the training dataset. A sample is as follows:

Sample Input 1:

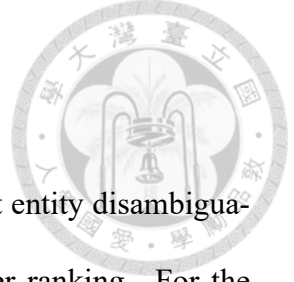
1.0 lux is the unit of illuminance for what unit?

Sample Output 1:

What unit is illuminated by 1.0 lux?

5.3 Experiment of Generation after Ranking Approach of KGQA on Noisy CKBQA

Since there are too many edges on a single node in the Knowledge Graph PKUmod, it is inefficient to implement a complex model as a ranker for the CKBQA dataset. Fortunately, the generator will not be limited by this. As a result, we want to know if the strategy generated after ranking is useful for the Noisy dataset and with a simple structure ranker. In this study, we designed a relation extraction experiment to compare the performance of different methods on the Noisy Knowledge Graph Question Answering task.



5.3.1 Experiment setting

For the entity extraction we use ngram entity searching and bert entity disambiguation. In the relation extraction, we use the strategy generating after ranking. For the ranker, we reference a leading competitor’s text comparison method and build the ranker. For the generator, we use the pretrain version of t5-pegasus(Su, 2021), which is specified for Chinese.

5.3.2 Experiment Result

The experiment mainly included the following three settings: ranker-only, ranker + t5-pegasus-small, and ranker + t5-pegasus-base.

5.3.2.1 Ranker Result

In this setting, we only used the text-based Ranker to rank candidate relations. We observed the performance of the Ranker alone on the relation extraction task. From Table 5.4, we can see that it is difficult for the ranker to solve the question with many hops.

	1-hop	2-hop	3-hop	4-hop	5-hop	All
Precision	0.5909	0.4182	0.2421	0.0000	0.0000	0.5140
Recall	0.5850	0.4234	0.2395	0.0000	0.0000	0.5119
F1	0.5842	0.4127	0.2402	0.0000	0.0000	0.5080
ACC	0.5597	0.3884	0.2381	0.0000	0.0000	0.4850

Table 5.4: Performance Evaluation of Our Modified Ranker



5.3.2.2 Ranker + Generator (Pegasus T5-base)

In this setting, we combined the Ranker with the Pegasus T5-base generative model. First, the Ranker ranked the candidate relations and then selected the 20 most probable relations. The Pegasus T5-base generative model was used to refine these candidate relations. This setting aimed to show the performance of the Ranker combined with the Pegasus T5-base generative model.

	1-hop	2-hop	3-hop	4-hop	5-hop	All
Precision	0.7252	0.6439	0.3835	0.7500	0.0000	0.6789
Recall	0.7146	0.6631	0.4527	1.0000	0.0000	0.6828
F1	0.7147	0.6430	0.3723	0.8333	0.0000	0.6717
ACC	0.6855	0.5868	0.3333	0.5000	0.0000	0.6327

Table 5.5: Performance Evaluation of Ranker + Pegasus T5-base

5.3.2.3 Ranker + Generator (Pegasus T5-small)

In this setting, we combined the Ranker with the Pegasus T5-small generative model. Similarly, the Ranker ranked the candidate relations first, and the Pegasus T5-small generative model refined the top 20 candidate relations. This setting showed the performance of the Ranker combined with the Pegasus T5-small generative model, which was not as good as Pegasus T5-base but still showed improvement over the Ranker alone.

	1-hop	2-hop	3-hop	4-hop	5-hop	All
Precision	0.6338	0.4293	0.2619	0.1667	0.0000	0.5458
Recall	0.6232	0.4455	0.2857	0.5000	0.0000	0.5465
F1	0.6241	0.4264	0.2698	0.2500	0.0000	0.5395
ACC	0.5996	0.3967	0.2381	0.0000	0.0000	0.5124

Table 5.6: Performance Evaluation of Ranker + Pegasus T5-small

When the Ranker is combined with Pegasus T5-base, the precision, recall, and F1 scores at each hop have significantly improved. This indicates that the generative model

can effectively refine the candidate relations selected by the Ranker, thereby improving the accuracy and coverage of relation extraction. In this setting, the comprehensive indicator (ACC) reached 0.6327, demonstrating the strong performance of the Ranker and Pegasus T5-base combination.

Compared with Pegasus T5-base, the use of a smaller Pegasus T5-small as the generator shows a slight decrease in performance at each hop. However, it still outperforms the method of only using the Ranker in terms of precision, recall, and F1 scores. This demonstrates that the Pegasus T5-small generator still has a certain relation refinement ability, but may not reach the performance level of Pegasus T5-base in all cases.

Combining the above discussions, we can draw the following conclusions: In the task of relation extraction, combining the Ranker and generative models (such as Pegasus T5-base or Pegasus T5-small) can effectively improve performance. Among these two generative models, Pegasus T5-base performs better, but Pegasus T5-small is still competitive in some cases. These experimental results provide useful guidance for future KBQA research and demonstrate the effectiveness of the relation extraction strategy combining Ranker and generative models.



Chapter 6 Discussion and Analysis

In this Chapter we are going to discuss the performance of auto-generation of Math dataset and the feasibility of combine ChatGPT and our KGQA approach.

6.1 Analysis of Math Word Problem

Dataset	Type of Question	Formula
MathQA	12,101	10,721
MathQA + Aqua ¹	22,149	18,339
MathQA + Aqua ³	28,193	27,558

Table 6.1: Comparison of the diversity of each dataset (Number of different types of question and different formulas).

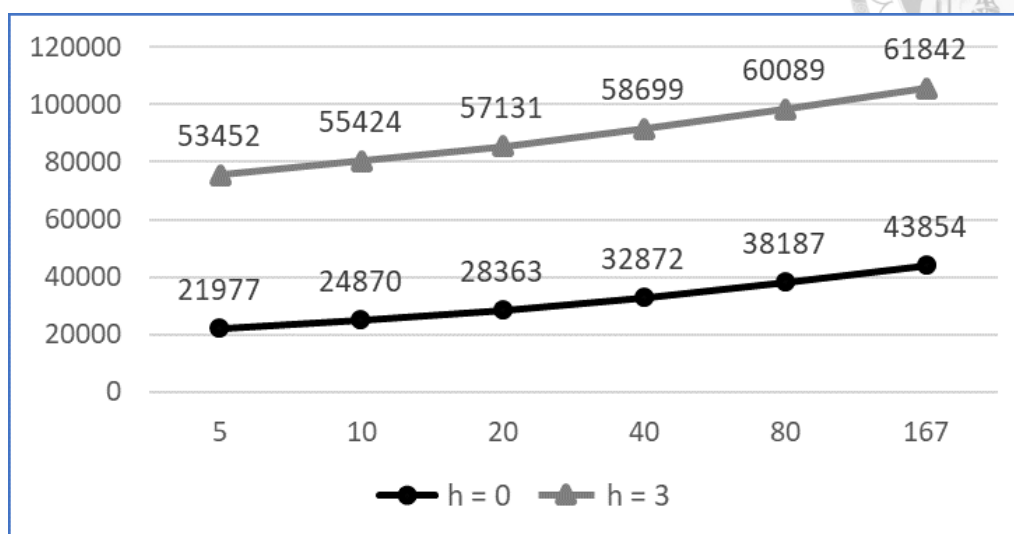
6.1.1 Analysis of Augmentation

In Table 6.1, we listed the number of different types of question and formula in each dataset. If two questions with different types, means those questions not only different in the numbers but also different in the structure.

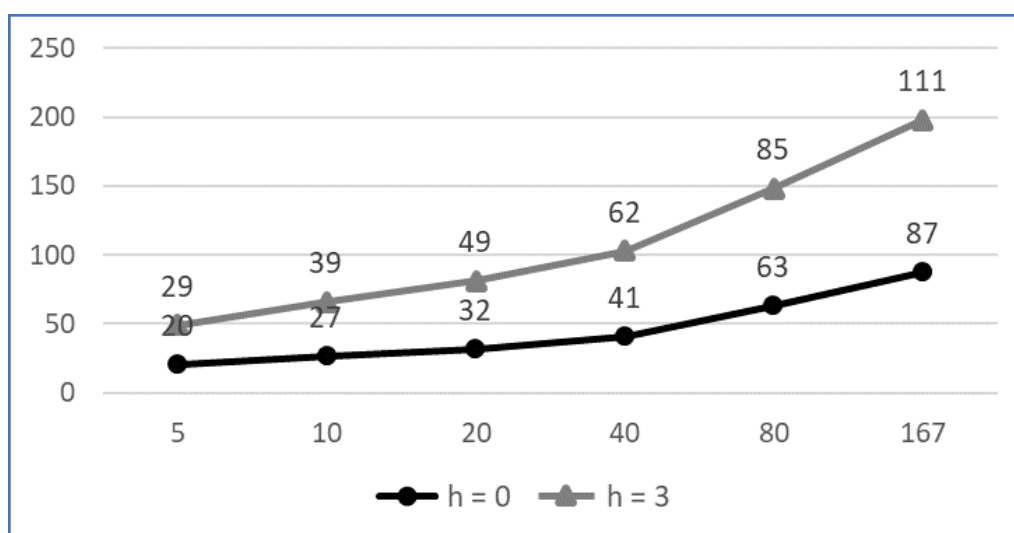
By the table, we could reasonably speculate that the solver train on the NDA datasets performed better because of the wider diversity of their data.



6.1.2 Analysis of Parameter h



(a) Verification on the training data (97,467 for all)



(b) Verification on the testing data (254 for all)

Figure 6.1: Verification on the training and test data. The horizon axis denotes the number of generated answers, while the vertical axis denotes the number of data being verified.

Figure 6.1 (a) shows an increasing trend with a high start point at both of the models trained on the original MathQA and that trained on the augmented MathQA because the training data of AQuA also contains parts of MathQA. The number of data that can be validated has a positive relation to the log of the number of generations. The data that can be validated by the model with NDA is more than the data that can be validated by

the model with only the original MathQA, which shows that the method can iteratively augment the dataset. To avoid the suspicion of the training data is the same as the test data (NDA model was trained on AQuA* + MathQA*), we also use the test split of AQuA to do the experiment.

As shown in Figure 6.1 (b), the model with NDA still performs better. We further compare the compositions of the question types and the distinct symbolic operations that are used to compose a program. In Table 6.1, we list the number of different types of questions and formulas in each dataset. With the augmented instances, the types of questions are increased from 12,101 to 28,193, and the types of operations are increased from 10,721 to 27,558, providing much more diverse and rich expressions for the model to learn to generate programs.

6.2 Case of Augmentation base Validation Combine with ChatGPT

In our ChatGPT version approach, we first utilize ChatGPT reverse generator to first generate questions based on the given program. Subsequently, we use the ChatGPT version generator to produce the final program(s). This section discusses the impact and feasibility of these two adaptations.

6.2.1 Question Generation by ChatGPT

We examine the impact of substituting the reverse generator with ChatGPT. Our analysis of several instances surprisingly revealed that questions generated by ChatGPT are

	Example
Readable Program	(AND cvg , computer videogame (JOIN cvg , computer videogame , versions Fighting Street))
Raw Data	fighting street is the version of which video game?
Augmented Question	what is the video game that has a version called fighting street?
Corrector Output	which video game has a version called fighting street?

Table 6.2: The sample of different versions of questions.

often superior to the original data, as the original data frequently contain words in an abnormal order. Table 6.2 shows a processed program where all unreadable words (e.g., entity id) are replaced with their human-friendly equivalents.

The questions derived from the dataset, the reverse generator, or the corrector all accurately convey the same meaning as the processed program. However, in natural language, questions typically start with wh-words, unlike the questions in the original dataset where the wh-word is positioned in the middle. In contrast, ChatGPT generates questions beginning with a wh-word when given the correct program, indicating that ChatGPT might produce better questions than the original dataset.

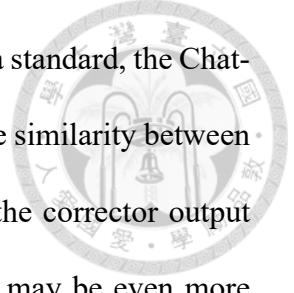
Hence, we implemented a corrector, whose role is described in Chapter 4.2.5. As shown in the examples, the corrector’s output closely resembles the reverse generator’s output when ChatGPT is used.

Reverse Generator Model	Compare Question	Bleu
ChatGPT	Corrector	0.6479
ChatGPT	Original	0.5864
T5	Original	0.5668

Table 6.3: Comparison of the reverse generator result.

To further investigate this phenomenon, we compared the similarity between questions generated by ChatGPT/T5 and those corrected by ChatGPT or sourced from the

original data. Table 6.3 shows that when using the original dataset as a standard, the ChatGPT version of the reverse generator outperforms the T5 version. The similarity between the output from the ChatGPT version of the reverse generator and the corrector output is also notably higher. These findings suggest that our architecture may be even more effective with the ChatGPT version of the reverse generator and corrector.



6.2.2 ChatGPT Generator (G)

In this section, we examine the feasibility of using ChatGPT as the generator. This adaptation significantly enhances the efficiency of the entire process, as it performs inline fine-tuning instead of the two steps of traditional fine-tuning.

Approach	Execution Ratio	F1(exec)
Ranker only	1.0000	0.7925
ChatGPT(G) (without Rev question)	0.0800	0.4225
ChatGPT(RevG + G)	0.6600	0.8878
ChatGPT(RevG + G) + Corrector	0.6300	0.9278

Table 6.4: Comparison of the ChatGPT and Ranker result under 100 instances.

Table 6.4 presents the Executable Ratio and F1(exec), defined similarly as in section 5.3. With the generator and reverse generator both powered by ChatGPT, the architecture achieves a F1(exec) score of 0.8878, significantly higher than the 0.7925 score achieved by using a ranker only. Despite some loss in the Execution Ratio due to the maximum token limit in GPT-3.5 (the generator can only view one program as valid), the version incorporating a corrector yields an impressive score of 0.9278. This suggests that our method may outperform any ranker in the GrailQA leaderboard in terms of reliability (F1(exec)). In certain instances where the ranker fails, the ChatGPT generator directly outputs: “I don’t know, it seems no answer in the reference.”, which means ChatGPT is

able to determine the meaning of the program under this approach.

These results are promising for the combination of large language models with our proposed approach.





Chapter 7 Conclusion

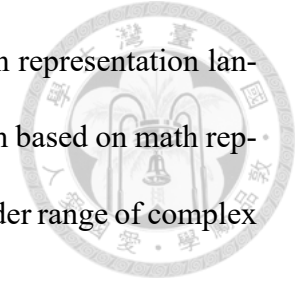
In this thesis, we have introduced several methods that exploit the characteristics of two distinct programs, thereby facilitating the creation of a superior model for addressing Math word problems and Knowledge Graph (KG) issues. Those methods can further help the tasks with the same features, which may even combine two methods if there's a program owns two features at the same time. Experimental evidence attests to their effectiveness.

To manage the scale issue of the training set involved in math word problem-solving tasks, we have proposed a novel numeracy data augmentation approach. This technique recycles to generate intermediate forms for extensive datasets annotated in an end-to-end style. We have also implemented symbolic verification to ensure the quality of the augmented data.

The experimental results not only validate the efficacy of these intermediate forms for learning but also indicate that our model significantly outperforms state-of-the-art models.

We have put forward a unique numeracy data augmentation approach, effective in increasing the quality of training instances. Our model's performance is state-of-the-art, as proven by the experimental results. Moreover, the results underscore the importance of incorporating reasoning process information when training models for math word problem-

solving. By utilizing data with minimal bias and data devoid of math representation language, we have reduced the cost of developing a math-solving system based on math representation language. Future work will extend our approach to a broader range of complex math problems.



In the KGQA task, we conducted several experiments. To devise a solution for the noisy dataset, we integrated a "generate after ranking" approach on the noisy Chinese data. The feasibility of this approach on the simple structure ranker has been demonstrated through experiments, leading to improved efficiency. To enhance the reliability of the models in providing Knowledge Graph answers, we have introduced an automatic question augmentation approach. This approach capitalizes on language models' ability to generate natural language questions, assisting the models in interpreting the programs and determining their correctness.

We have also investigated the potential of combining a Large Language model with this approach. Our results show that GPT3.5 (ChatGPT) harmonizes well with the introduced approach.

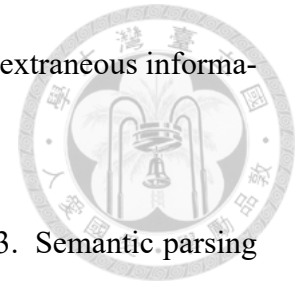
With the imminent release of the long context version of GPT3.5/4, it is anticipated that our work could be further expanded in various settings, leading to improved results.



References

- Aida Amini, Saadia Gabriel, Shanchuan Lin, Rik Koncel-Kedziorski, Yejin Choi, and Hannaneh Hajishirzi. 2019. Mathqa: Towards interpretable math word problem solving with operation-based formalisms. 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 2357–2367.
- Mattia Atzeni, Jasmina Bogojeska, and Andreas Loukas. 2021a. Sqaler: Scaling question answering by decoupling multi-hop and logical reasoning. Advances in Neural Information Processing Systems, 34:12587–12599.
- Mattia Atzeni, Jasmina Bogojeska, and Andreas Loukas. 2021b. Sqaler: Scaling question answering by decoupling multi-hop and logical reasoning. Advances in Neural Information Processing Systems, 34:12587–12599.
- Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. 2007. Dbpedia: A nucleus for a web of open data. In The Semantic Web: 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007+ ASWC 2007, Busan, Korea, November 11-15, 2007. Proceedings, pages 722–735. Springer.

Yefim Bakman. 2007. Robust understanding of word problems with extraneous information. [arXiv preprint math/0701393](https://arxiv.org/abs/math/0701393).



Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic parsing on freebase from question-answer pairs. In Proceedings of the 2013 conference on empirical methods in natural language processing, pages 1533–1544.

Daniel G. Bobrow. 1964. Natural language input for a computer problem solving system. Technical report, USA.

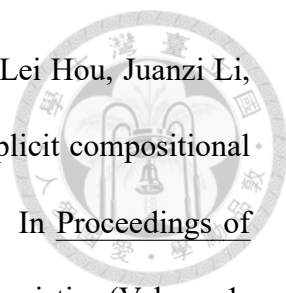
Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. 2008. Freebase: a collaboratively created graph database for structuring human knowledge. In Proceedings of the 2008 ACM SIGMOD international conference on Management of data, pages 1247–1250.

Antoine Bordes, Nicolas Usunier, Sumit Chopra, and Jason Weston. 2015. Large-scale simple question answering with memory networks. [arXiv preprint arXiv:1506.02075](https://arxiv.org/abs/1506.02075).

Antoine Bordes, Jason Weston, and Nicolas Usunier. 2014. Open question answering with weakly supervised embedding models. In Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2014, Nancy, France, September 15-19, 2014. Proceedings, Part I 14, pages 165–180. Springer.

Diane J Briars and Jill H Larkin. 1984. An integrated model of skill in solving elementary word problems. Cognition and instruction, 1(3):245–296.

Qingqing Cai and Alexander Yates. 2013. Large-scale semantic parsing via schema matching and lexicon extension. In Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 423–433.



Shulin Cao, Jiaxin Shi, Liangming Pan, Lunyiu Nie, Yutong Xiang, Lei Hou, Juanzi Li, Bin He, and Hanwang Zhang. 2022. Kqa pro: A dataset with explicit compositional programs for complex question answering over knowledge base. In Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 6101–6119.

Kezhen Chen, Qiuyuan Huang, Hamid Palangi, Paul Smolensky, Kenneth D Forbus, and Jianfeng Gao. 2020. Mapping natural-language problems to formal-language solutions using structured neural representations. In Proc. of ICML, volume 2020.

Xinyun Chen, Chen Liang, Adams Wei Yu, Denny Zhou, Dawn Song, and Quoc V Le. 2019. Neural symbolic reader: Scalable integration of distributed and symbolic representations for reading comprehension. In International Conference on Learning Representations.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805.

Li Dong, Furu Wei, Ming Zhou, and Ke Xu. 2015. Question answering over freebase with multi-column convolutional neural networks. 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 260–269.

Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. 2019. DROP: A reading comprehension benchmark requiring discrete reasoning over paragraphs. In Proc. of NAACL.

Mohnish Dubey, Debayan Banerjee, Abdelrahman Abdelkawi, and Jens Lehmann. 2019.

Lc-quad 2.0: A large dataset for complex question answering over wikidata and dbpedia. In The Semantic Web–ISWC 2019: 18th International Semantic Web Conference, Auckland, New Zealand, October 26–30, 2019, Proceedings, Part II 18, pages 69–78. Springer.

Mor Geva, Ankit Gupta, and Jonathan Berant. 2020. Injecting numerical reasoning skills into language models. In ACL.

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

Steve Harris and Andy Seaborne. 2013. SPARQL 1.1 query language. W3C Recommendation 21 March 2013, World Wide Web Consortium.

Charles Hinson, Hen-Hsen Huang, and Hsin-Hsi Chen. 2020. Heterogeneous recycle generation for Chinese grammatical error correction. In Proceedings of the 28th International Conference on Computational Linguistics, pages 2191–2201, Barcelona, Spain (Online). International Committee on Computational Linguistics.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. Neural computation, 9(8):1735–1780.

Mohammad Javad Hosseini, Hannaneh Hajishirzi, Oren Etzioni, and Nate Kushman. 2014. Learning to solve arithmetic word problems with verb categorization. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), pages 523–533.

Danqing Huang, Jing Liu, Chin-Yew Lin, and Jian Yin. 2018a. Neural math word problem

solver with reinforcement learning. In Proceedings of the 27th International Conference on Computational Linguistics, pages 213–223.



Danqing Huang, Jin-Ge Yao, Chin-Yew Lin, Qingyu Zhou, and Jian Yin. 2018b. Using intermediate representations to solve math word problems. In Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 419–428.

Sarthak Jain. 2016. Question answering over knowledge base using factual memory networks. In Proceedings of the NAACL student research workshop, pages 109–115.

Daniel Keysers, Nathanael Schärli, Nathan Scales, Hylke Buisman, Daniel Furrer, Sergii Kashubin, Nikola Momchev, Danila Sinopalnikov, Lukasz Stafiniak, Tibor Tihon, et al. 2019. Measuring compositional generalization: A comprehensive method on realistic data. In International Conference on Learning Representations.

Yunshi Lan and Jing Jiang. 2020. Query graph generation for answering multi-hop complex questions from knowledge bases. Association for Computational Linguistics.

Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. 2017. Program induction by rationale generation: Learning to solve and explain algebraic word problems. In Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 158–167.

Thang Luong, Ilya Sutskever, Quoc Le, Oriol Vinyals, and Wojciech Zaremba. 2015. [Addressing the rare word problem in neural machine translation](#). 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 11–19, Beijing, China. Association for Computational Linguistics.

OpenAI. 2021. Chatgpt: Large-scale language model. <https://openai.com/research/chatgpt>. Accessed: July 3, 2023.



Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21(1):5485–5551.

Srinivas Ravishankar, June Thai, Ibrahim Abdelaziz, Nandana Mihidukulasooriya, Tahira Naseem, Pavan Kapanipathi, Gaetano Rossiello, and Achille Fokoue. 2021. A two-stage approach towards generalization in knowledge base question answering. *arXiv preprint arXiv:2111.05825*.

Subhro Roy, Tim Vieira, and Dan Roth. 2015. Reasoning about quantities in natural language. *Transactions of the Association for Computational Linguistics*, 3:1–13.

Apoorv Saxena, Aditay Tripathi, and Partha Talukdar. 2020. Improving multi-hop question answering over knowledge graphs using knowledge base embeddings. In *Proceedings of the 58th annual meeting of the association for computational linguistics*, pages 4498–4507.

Yiheng Shu, Zhiwei Yu, Yuhan Li, Börje F Karlsson, Tingting Ma, Yuzhong Qu, and Chin-Yew Lin. 2022. Tiara: Multi-grained retrieval for robust question answering over large knowledge bases. *arXiv preprint arXiv:2210.12925*.

Jianlin Su. 2021. [T5 pegasus - zhuiyai](#). Technical report.

Yawei Sun, Lingling Zhang, Gong Cheng, and Yuzhong Qu. 2020. Sparqa: skeleton-based semantic parsing for complex questions over knowledge bases. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 8952–8959.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. Advances in neural information processing systems, 27.

Dung Thai, Srinivas Ravishankar, Ibrahim Abdelaziz, Mudit Chaudhary, Nandana Mihindukulasooriya, Tahira Naseem, Rajarshi Das, Pavan Kapanipathi, Achille Fokoue, and Andrew McCallum. 2022. Cbr-ikb: A case-based reasoning approach for question answering over incomplete knowledge bases. arXiv preprint arXiv:2204.08554.

Priyansh Trivedi, Gaurav Maheshwari, Mohnish Dubey, and Jens Lehmann. 2017. Lcquad: A corpus for complex question answering over knowledge graphs. In The Semantic Web–ISWC 2017: 16th International Semantic Web Conference, Vienna, Austria, October 21-25, 2017, Proceedings, Part II 16, pages 210–218. Springer.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In Advances in neural information processing systems, pages 5998–6008.

Denny Vrandečić and Markus Krötzsch. 2014. Wikidata: a free collaborative knowledge-base. Communications of the ACM, 57(10):78–85.

Xu Wang, Shuai Zhao, Jiale Han, Bo Cheng, Hao Yang, Jianchang Ao, and Zhenzi Li. 2020. Modelling long-distance node relations for kbqa with global dynamic graph. In Proceedings of the 28th International Conference on Computational Linguistics, pages 2572–2582.

Xi Ye, Semih Yavuz, Kazuma Hashimoto, Yingbo Zhou, and Caiming Xiong. 2022. Rng-kbqa: Generation augmented iterative ranking for knowledge base question answering. In Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 6032–6043.

Wen-tau Yih, Matthew Richardson, Christopher Meek, Ming-Wei Chang, and Jina Suh.

2016. 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.

Yuyu Zhang, Hanjun Dai, Zornitsa Kozareva, Alexander Smola, and Le Song. 2018. Variational reasoning for question answering with knowledge graph. In Proceedings of the AAAI conference on artificial intelligence, volume 32.