Domain-specific QA with Large Language Models over Knowledge Graphs

Nikita Peleshatyi

4th Year Project Report
Software Engineering
School of Informatics
University of Edinburgh

2025

Abstract

This skeleton demonstrates how to use the `infthesis` style for undergraduate dissertations in the School of Informatics. It also emphasises the page limit, and that you must not deviate from the required style. The file `skeleton.tex` generates this document and should be used as a starting point for your thesis. Replace this abstract text with a concise summary of your report.

Research Ethics Approval

This project was planned in accordance with the Informatics Research Ethics policy. It did not involve any aspects that required approval from the Informatics Research Ethics committee.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Nikita Peleshatyi*)

Acknowledgements

I thank Doctor-in-making Max Ries for his continued support and advice throughout this project, our discussions taught me a lot about formalising my thinking and reasoning. I also thank Dr. Sohan Seth for valuable feedback and support.

Above all, I would like to thank my family for granting me such an invaluable education opportunity culminating in this very dissertation marking the most valuable investment one can make. Education.

Table of Contents

1 Introduction **1**

1.1 Motivation . . . 1

1.2 Aims and Contributions . . . 2

1.3 Project Overview . . . 2

2 Background and Related Work **3**

2.1 Brief Introduction to QA . . . 3

2.2 Language Modelling . . . 4

2.3 Retrieval Augmented Generation . . . 7

3 Methodology **11**

3.1 Data and Models . . . 11

3.1.1 Dataset Selection and Verification . . . 11

3.1.2 Model Selection . . . 13

3.1.3 Accuracy Specification . . . 14

3.2 Baseline Performance . . . 14

3.2.1 Closed-Book Performance . . . 15

3.2.2 Perfect-Path Performance . . . 15

3.3 Relevance-Based Graph Pruning . . . 16

3.3.1 Task Formulation . . . 16

3.3.2 BM25 . . . 18

3.3.3 S-BERT . . . 18

3.4 Prediction-Based Graph Traversal . . . 20

3.4.1 Link Prediction and Retrieval . . . 21

3.4.2 Tail pruning . . . 21

3.4.3 Reasoning . . . 23

4 Results **24**

4.1 Baseline and Previous Studies . . . 24

4.2 Relevance-Based Methods . . . 26

4.3 ToG-LP . . . 31

5 Discussion **33**

5.1 Baseline . . . 33

5.2 Relevance-Based Methods . . . 34

5.3 ToG-LP . . . 35

5.4 Case Study: CWQ Question Types 36

6 Conclusions **38**
6.1 Summary 38
6.2 Challenges and Limitations 40
6.3 Future Work 40

Bibliography **41**

A Supplementary material **47**
A.1 Temperature 47
A.2 Dataset verification 48
A.3 Prompts 48

B Results Tables **49**
B.1 FBQA 50

Chapter 1

Introduction

1.1 Motivation

Building intelligent entities that can perceive, understand and interact with the world around us has been a long-standing goal of the field of Artificial Intelligence since its inception [1]. The proliferation of the Internet and the emergence of neural technologies such as LLMs have opened a number of new prospective applications in automation, including improved question answering.

In many fields, such as medicine, law, and engineering, practitioners face similar challenges of accessing, navigating and processing vast amounts of data. And it keeps increasing (get some number of the amount of information) (maybe some numbers of the number of experts needed and available)

Domain-specific intelligent system can significantly improve productivity by facilitating exploration and decision-making processes. (find some backing)

1.2 Aims and Contributions

The goal of this project is to leverage the language comprehension capabilities of LLMs and extend them with domain-specific knowledge in specialised fields where information retrieval is beneficial. In doing so, our aim is to get an understanding of what automated questions answer is, how it is done, and how we can develop transparent and explainable reasoning techniques for answering questions on specialised knowledge graphs.

The main contributions of this project are as follows.

- Compiled a comprehensive overview of modern techniques for automated question answering. Focused on one subfield in more detail and performed quantitative analysis through experiments.
- Selected several appropriate QA datasets with corresponding knowledge graphs. Each set of question was verified to be consistent with the knowledge base. Incomplete sets were augmented with existing data from other sources.
- Developed a framework for conducting and evaluating experiments. Modular design allows models, heuristics, datasets and knowledge bases to be easily replaced.
- Established baseline performance on Mistral7B to get expected lower and upper boundaries for evaluating downstream performance. Compared the results with relevant existing State-of-the-Art methods.
- Investigated two relevance-based context scoring heuristics: BM25 and S-BERT, respectively. Analysed performance with different settings and the resulting benefits and trade-offs.
- Building on previous work, modified an autonomous graph traversal algorithm for efficiency gains and compared the results with the two heuristics.
- Identified challenges, limitations, and potential directions for future work.

1.3 Project Overview

The report is structured as follows: *Chapter 2, Background and Related Work*, introduces the field of automated question answering, traditional approaches, what the recent developments are and how they can be improved with modern neural network-based approaches. *Chapter 3, Methodology*, describes the resources available, data and models, their configurations, proposed relevance-based graph exploration heuristics and a novel graph traversal algorithm, as well as our performance-optimised modification to it. *Chapter 4, Results*, presents the final obtained results comparing them with the existing State-of-the-Art. *Chapter 5, Discussion*, examines the results in more detail, discussing their origins, as well as their relative benefits and trade-offs. *Chapter 6, Conclusions*, summarises the findings, highlights limitations and contains recommendations for future work.

Chapter 2

Background and Related Work

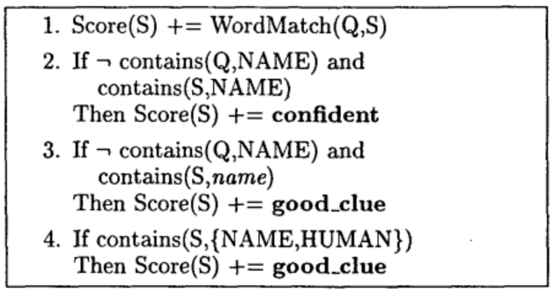
2.1 Brief Introduction to QA

Question Answering (QA) is a specialised area of Information Retrieval (IR)[2] that is concerned with providing relevant answers in response to questions provided as natural language, i.e. with no particular format as shown in Figure 2.1. Unlike conventional IR systems, which return ranked list of documents that may contain relevant information, the objective of QA systems is to extract and present the answer to a given query from a collection of documents [2].

The official language of Austria is what?

Figure 2.1: Simple natural language query.

Traditional QA systems rely on rule-based approaches with a set of handcrafted rules to handle certain types of queries [2, 3]. First, the question type is identified based on keywords (What, When, Who), and a corresponding scoring algorithm is applied to rank individual sentences. The procedure often involves parsing rules with first-order logic or selection (i.e. if-then) to convert queries and statements to a format that the algorithm can process [3]. An example of such algorithm is shown in Figure 2.2.



```
1. Score(S) += WordMatch(Q,S)
2. If ¬ contains(Q,NAME) and
      contains(S,NAME)
   Then Score(S) += confident
3. If ¬ contains(Q,NAME) and
      contains(S,name)
   Then Score(S) += good_clue
4. If contains(S,{NAME,HUMAN})
   Then Score(S) += good_clue
```

Figure 2.2: Example of a rule-based sentence scoring algorithm for a WHO type question. Scores are assigned based on literal word and question type specific feature match. Allows partial match scores and has several variable parameters. Figure taken from [3]

This approach requires an explicit definition of lexical, grammatical, and occasional domain-specific rules for decomposing the query before it could be matched against a knowledge base to extract the answer [3]. While these systems offer high explainability and transparency, their effectiveness is entirely dependent on the variety and quality of the constructed ruleset [4]. This approach lacks adaptability to novel types of questions, is not robust to minor variations in input data and requires human domain experts to construct the rules manually [4].

Recent advances in machine learning and deep learning facilitated the transition to a new paradigm, which addresses these limitations and offers a more flexible approach to automated question answering [5]. By learning latent representations of words and sentences [6], they capture semantic and syntactic relationships implicitly [7]. Rule-based QA systems typically require at least 3 stages: question classification, information retrieval, and answer extraction, where each step has a degree of rigidity in hand-crafted algorithms [4]. The flexibility and versatility of neural network-based language models offer a way to enrich, combine, or rethink the steps in traditional approaches. A key advantage of such language models is their intrinsic ability to process and extract information from unstructured text [8], eliminating the need for manual rule-based processing mechanisms, making them functional agents for answering natural language queries.

2.2 Language Modelling

Advances in hardware, software and the increasing availability of large-scale data, largely driven by the proliferation of the Internet, have facilitated the evolution of natural language processing (NLP) from symbolic rule-based systems to statistical models, such as N-grams and Hidden Markov Models, and subsequently to deep neural networks.

N-gram models (presented in Figure 2.3a) estimate the probability of a word occurring given the preceding $N-1$ words, and are based on counts of word co-occurrences in a corpus, which makes it quick and cheap to train. However, it is prone to generate unreliable or zero probabilities due to data sparsity. The model also uses the assumption that any given word depends on a fixed-length preceding context, which limits the ability to robustly capture various linguistic dependencies [9].

Hidden Markov Models, in contrast, introduce latent state transitions and separate them from observable emitted words and are also based on counts. This allows for higher expressivity but has the same drawbacks as N-gram models [10]. Moreover, HMMs need explicitly labelled training data to account for latent states, as shown in Figure 2.3b.

$$P(\text{word} \mid \text{previous } N-1 \text{ words}) = P(w_1,\ldots,w_n)$$

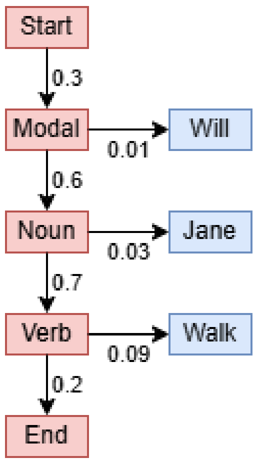
$$\approx \prod_{i=1}^{n} P(w_i \mid w_{i-(N-1)},\ldots,w_{i-1})$$

where

$$P(w_i \mid w_{i-(N-1)},\ldots,w_{i-1}) = \frac{C(w_{i-(N-1)},\ldots,w_i)}{C(w_{i-(N-1)},\ldots,w_{i-1})}$$

and $C(\cdot)$ is the count of N-grams in a corpus

(a) N-gram model formula definition



(b) Example of a Hidden Markov Model. Latent states (parts of speech) are in red, emitted words are in blue, probabilities represent likelihood of transition.

Figure 2.3: Statistical approaches to language modelling

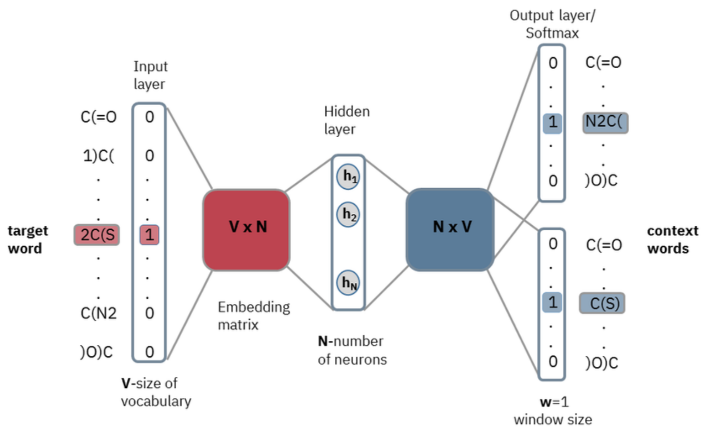


Figure 2.4: Skip-gram model architecture. The goal is to predict nearby context of a given target word. Size of the context is determined by the window size w. In this example, $w = 1$, so two words will be predicted (one on the left and one on the right). V is the vocabulary size, N is the number neurons (size of the embedding vector). The weight matrix $V \times N$ converts input to a dense vector. The matrix $N \times V$ is used to predict the probability of context words. This example is taken from [11].

The transition to more advanced models, such as neural networks, addressed these limitations by enabling machine learning algorithms to learn feature representations and their interdependencies as vectors embedded in a latent space [12]. Word2Vec is a method for computing a distributed representation of words [6]. It builds on the intuition of N-grams that words occurring together in some context are statistically more dependent [13], but replaces discrete counting with vector representation, which is learnt from random state using the Skip-gram model architecture [6] (Figure 2.4).

Improved word-level understanding allows to better capture semantic similarity and address one of the limitations of rule-based QA approach. However, it provides limited understanding of sentence-level structure due to the Skip-gram's architectural limitations of the context size [6]. Recurrent Neural Networks (RNNs), such as LSTM[14], leverage word vector representation to model dependencies that theoretically span sentences and paragraphs by updating the hidden state at every step of the sequence. Need for sequential processing limits efficient scalability. A single updated state that pass information at arbitrarily long distances suffers from the vanishing gradient problem that restricts information retention capabilities [14].

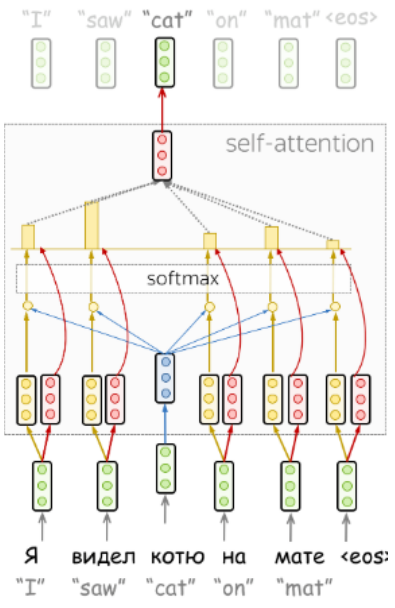


Figure 2.5: Self-attention mechanism. Each token in sequence attends to the entire sequence. In this case, the vectors for tokens are different depending on their role: blue for Query token, yellow for Key token and red for Value. Diagram taken from NLP course

Self-attention (visualised in Figure 2.5) is an alternative mechanism for processing sequential data, in which every token attends to every other token in a sequence updating the representation to a more context-specific one [7]. It comes with a higher computational cost since full attention requires n^2 operations, where n is the length of a sequence, unlike RNN, where the number of operations grows linearly with n. However, different configurations can optimise performance as visualised in Figure 2.6.

Two key benefits of using self-attention are amenity to parallelisation and more efficient approach to modelling linguistic dependencies [7]. Since every token can attend to the surrounding context independently, the process can be parallelised and completed in any arrangement, preserving the notion of order with positional encodings [7]. Because every token attends to every other, the maximum path length between any two tokens is $O(1)$, or at most $O(n/k)$ in restricted configurations (Figure 2.6), which makes dependency modelling easier [7].

Notable robustness and effectiveness of transformer-based language models had spurred more research into different architectures, sizes, and configurations of these models [16].

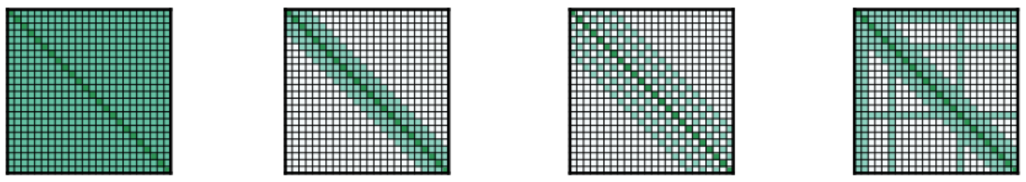

(a) Full n^2 attention (b) Sliding window attention (c) Dilated sliding window (d) Global+sliding window

Figure 2.6: Attention mechanism and its different configurations. Figure taken from [15].

Further scaling of model sizes and volumes of training data had coined the term Large Language Models (LLMs). LLMs exhibit a remarkable ability to follow instructions without explicit instructions, which was described as an ”emergent capability” and tends to improve at scale [17]. Moreover, providing several examples inside the prompt allowed LLMs to achieve results comparable with those of the State-of-the-Art at the time on many NLP tasks, including QA [18].

Although the nature of these capabilities is subject to further research, they are already useful as agents that can interpret complex tasking prompts, reason about them, and can even be extended to perform actions [19], making them a perfect replacement for rule-based QA systems.

2.3 Retrieval Augmented Generation

As the models grow in size and the amount of data they are trained on, their ability to recall factual knowledge and reason about it improves [17, 20, 21]. However, due to the lack of precise mechanistic interpretability in how this knowledge is stored and retrieved upon inference, these models effectively become black boxes producing coherent text [22]. Training objectives cause generated sentences to conform to grammatical norms, rather than logic [20]. While LLMs produce SOTA-comparable results on QA benchmarks with prompts and examples [17, 18, 19], the interpretability and explainability of those answers is not faithful as verification via mechanistic interpretability is not well-studied [22].

A popular approach to overcome this limitation is to leverage language comprehension capabilities and augment them with transparent information retrieval heuristics [20, 21]. In general, there are 2 paradigms to augmenting a language model with additional knowledge to that it was trained on: architecture-level and prompt-level [20, 21]. They can further be categorised into the 3 main methods of doing so, each with its own set of benefits and trade-offs (The differences are summarised in Table 2.1).

Retraining and fine-tuning provide the desired specialisation for domain-specific applications, such as engineering, law, or medicine, but require substantial amounts of computational capacity, increasing the cost, and a vast amount of training data, which there may not be enough of to attain the same reasoning capabilities. Given a small specialised dataset, the model is very likely to overfit to that set and cause more hallucinations [23]. More importantly, the key limitation of the 2 methods is that the model still remains a black-box, providing no explanation to how the answer was generated.

Paradigm	**Architecture-level**		**Prompt-level**
Method	**Training**	**Fine-Tuning**	**RAG**
Benefits	• Full customization • Domain specific • No bias • Data safety	• Integrates knowledge directly • Fast & Cheap • Data safety	• Dynamic knowledge injection • Explainable • No training required • Modular
Drawbacks	• Requires compute • Needs a lot of data • Lacks generalization • Black-box	• Requires quality data • Prone to overfitting • Loses generalization • Black-box	• Relies on retrieval system • Relies on knowledge quality • Latency

Table 2.1: Paradigms and methods of integrating external knowledge into pre-trained language models. The table summarises benefits and drawbacks of each one.

Alternatively, recent experiments show that ad-hoc methods, such as retrieval augmented generation (RAG) [24], prove to be effective in reasoning over a specialised collections of documents. It is a method that integrates external knowledge with the prompt and does not require any modifications to the original model [17, 18]. The model is used as an agent to combine the retrieved context and to answer the question [24]. Within the method, there are several approaches to interact with and integrate the external source of knowledge depending on the structure of the data. The 2 most prominent ones are Text RAG and Graph RAG respectively, the differences are summarised in Table 2.2.

Basic Text RAG is performed on unstructured documents, split into chunks, and put into an index for efficient retrieval. It has clearly separated modules (agent, retriever, ranker) that can be easily updated and replaced. It scales well to large collections with vector indexing (HNSW, FAISS), allows the main reasoning agent to be easily replaced, and provides clear referencing of the context it used to generate the answer. The key limitation of Text RAG is that it relies entirely on semantic similarity, missing hierarchical connections in data that are crucial in certain domains, significantly limiting their performance at handling complex queries. The semantic similarity in this case is also purely mathematical based on previous methods that represent the meaning in latent space. The problem is visually described in Figure to make (A-D retrieves A and D but misses A-**B**-**C**-D) .

	Text RAG	**Graph RAG**	**Hybrid RAG**
Mechanism	Retrieves chunks of unstructured data ranked by semantic similarity	Knowledge is explored via queries on structured relations	Blend of chunk retrieval, semantic ranking and structured relations querying.
Benefits	• Modular • Better explainability • Efficient at scale	• Highly precise & transparent	• Combines structure and nuance
Drawbacks	• Relies on semantic similarity • Little structure • Unable to process complex queries • Storage & compute requirements	• Hard to construct a graph • Hard to differentiate domain from general knowledge • Requires exploration heuristic	• Complexity in management and usage • Compute & storage • Requires alignment

Table 2.2: The table summarises mechanism, benefits and drawbacks of the most popular approaches to RAG: TextRAG, GraphRAG and their hybrid.

Graph RAG is performed on a structure called knowledge graph (example shown in Figure 2.7) derived from the unstructured data. A knowledge graph is a structured representation of concepts, facts, and relationships between them. A knowledge graph is defined as: $G = (V, E)$, where V is the set of vertices representing facts and concepts, and E is the set of relations connecting entities. The relationship is formally defined as a triplet in the form of: (h, r, t), where $h, t \in V$ are head and tail entities, respectively, and $r \in E$ is the relationship between them.

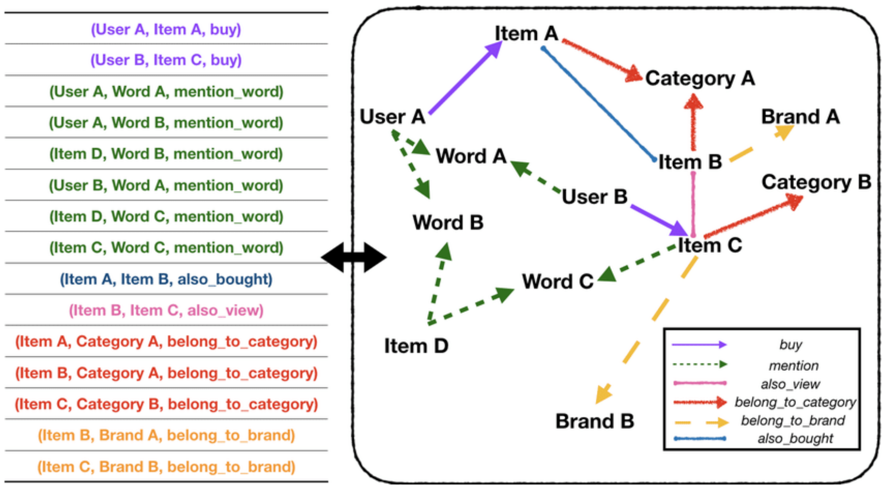

(User A, Item A, buy)
(User B, Item C, buy)
(User A, Word A, mention_word)
(User A, Word B, mention_word)
(Item D, Word B, mention_word)
(User B, Word A, mention_word)
(Item D, Word C, mention_word)
(Item C, Word C, mention_word)
(Item A, Item B, also_bought)
(Item B, Item C, also_view)
(Item A, Category A, belong_to_category)
(Item B, Category A, belong_to_category)
(Item C, Category B, belong_to_category)
(Item B, Brand A, belong_to_brand)
(Item C, Brand B, belong_to_brand)
Item A
Category A
User A
Brand A
Word A
Item B
Category B
User B
Word B
Item C
Word C
Item D
Brand B
buy
mention
also_view
belong_to_category
belong_to_brand
also_bought

Figure 2.7: Example of a knowledge graph and corresponding knowledge triplets in the context of product recommendations. Figure taken from [25]

Unlike TextRAG, GraphRAG operates on well-structured data and offers a high degree of faithfulness in the answers generated with it. Some more modern approaches combine the two into HybridRAG achieving the best of both worlds, however it is a complex process and requires alignment and management overhead [26].

A knowledge graph itself, on the other hand, poses difficulty in construction and is a distinct research problem on its own. Both approaches suffer from the context-size limits of LLMs. Due to polynomial increase in complexity of computing attention, processing long input sequence becomes inefficient without optimizations, which cause hallucinations and degrade the ability to follow instructions [27]. This requires narrowing down the context provided to the LLM. In the case of TextRAG it is done by re-ranking with a b relevance scoring function [24]. GraphRAG, while supporting relevance scoring, enables algorithmic traversal of the knowledge graph for retrieving and reasoning about the knowledge. Theoretically, this makes the system more transparent and introduces more user-accessible settings for knowledge retrieval[1].

Following the original objectives of achieving transparent and explainable reasoning, as well as studying the landscape of data available, in this work we focus on Graph RAG and knowledge graph exploration for answering questions.

[1]Given an interface, the user can preselect or direct to the areas of the graph, which are relevant.

Chapter 3

Methodology

3.1 Data and Models

3.1.1 Dataset Selection and Verification

Domain-specific QA tasks require a highly relevant and complete knowledge base consisting of expert knowledge, technical documentation, and other proprietary information, which is not available online due to licensing and commercial restrictions. Following knowledge graph surveys [20, 21, 28] and studying the landscape of publicly available data, the following conclusions can be drawn:

Knowledge bases, such as MMedC[29], and EU/UK legislation[30] provide highly relevant data, which is useful for answering questions from the derivative datasets [31, 32], however, the knowledge is provided as unstructured free-format text, making it unsuitable for the purposes of this project without specifically constructing a knowledge graph first, which is unfeasible given the time constraints and the need for manual quality verification.

On the other hand, the explicitly constructed knowledge graphs PharmKG[33] and SoccerKG[34] are perfect for downstream applications but focus solely on the knowledge graph extraction, lacking the respective QA datasets. Constructing a dataset was not a viable option either, as that would require manual expert verification of ground answers and alignment with the graph.

The trade-off is between variety in structure and completeness of domain knowledge and the availability of suitable datasets to use with this knowledge. Both limitations become more extreme with the narrower specificity of the domain. However, there are several less specialized sources of data that satisfy both criteria.

Freebase

Freebase (FB) [35] is an open source collaboratively constructed general knowledge graph that has been continuously populated since 2007 until it was acquired by Google in 2015, after which the API was shut down and compatible knowledge had been merged with WikiData[1]. A lot of research had already been done, and many QA datasets were constructed based on Freebase. For this reason, most authors try to provide the relevant subsets of the graph they used. However, they are not always available (broken/missing links, second-hand reuploads) and therefore the data needs to be manually verified for consistency.

Recognizing it's significant impact on research, Google is still hosting the last Freebase dump [36] containing **49 million entities** and **1.9 billion triples**, amounting to 250GB in size. This dump was used to complete the missing subgraphs that were provided for the following QA datasets:

- **FreebaseQA** (FBQA) [37] is an open-domain question dataset with trivia-style questions which do not require complex reasoning, only fact retrieval. It is created from several previous distinct QA datasets and the FB itself, which makes it very linguistically diverse, perfect for LLM-based interpretation and answering. It contains over 54k machine-generated and human-verified question-answer pairs.
- **ComplexWebQuestions** (CWQ) [38] is a dataset of complex questions that require the ability to retrieve several data points and reason over them giving comparative, superlative or conjunctive answers. Unlike FBQA, CWQ focuses more on complex question types, rather than simple fact retrieval. Also machine-generated and human-verified (with paraphrasing), it contains over 34.6k examples along with supporting reasoning metadata.

WikiMovies

Originally a QA dataset, WikiMovies [39] also provides a relevant Knowledge Graph with **16 million entities** and **66 million triplets** extracted from Wikipedia articles and OMDb[2]. Although substantially smaller than Freebase, it is high-quality domain-specific knowledge and has several curated QA datasets associated with it. The most suitable one:

- **MetaQA** [40] is a large dataset of almost 400k questions involving both single- and multi-hop reasoning over the WikiMovies knowledge graph. Generated from a set of human-authored templates varying from 15 to 21 question types depending on the complexity (1-3 hops) and verified with negative sampling.

As some datasets are actively used as challenges, where test set answers were not available, development sets were used instead.

Each dataset was verified to be consistent (cite appendix table)[] with the respective knowledge base to make sure the relevant topic and answer entities are present and there exists at least one path between them. Unanswerable entries were filtered out. Due to thesis-time limitations and large computational costs for LLM inference, all test sets were reduced to 1000 entries each using a fixed random seed for reproducibility and are treated as representative of the original test sets.

[1]https://wikidata.org/

[2]https://www.omdbapi.com/

	FBQA	CWQ	MetaQA
Domain	Trivia-style general knowledge questions	Complex reasoning	Movie-specific questions
Example	Which country hosted the 1936 Summer Olympic Games?	Which Miley Cyrus movies were released after 12/10/2003?	What movies was Marianne Faithfull an actor in?
Test set size	4k	2.2k	15k

Table 3.1: Summary of the 3 datasets used in this study. FreebaseQA, ComplexWebQuestions and MetaQA, respectively.

3.1.2 Model Selection

The experiments were done on a GPU compute cluster with 2x24GB VRAM (Nvidia GeForce RTX 3090), 64GB RAM, 32 core 3.6GHz CPU (AMD Genoa EPYC). Given the setup, to achieve optimal performance and parallel inference on both GPUs, the main model + input data + other models were chosen to fit entirely on each unit.

Main Model

Most unquantized LLMs today use half-precision (float16) to store the weights. This means that for each model parameter, you need 16 bits or 2 Bytes of storage and compute capacity. This equates to 2GB per 1 Billion parameters. Given the ever-increasing range of LLMs available [16] varying in size and pre-training data, the optimal candidate size was to have between 5 and 10 Billion parameters.

As the project originates from a company proposal, preference was given to the models whose licence had unrestricted research and commercial usage, such as MIT or Apache 2.0. The most suitable candidate is Mistral7B [41], since it has a suitable license, open source weights, is fine-tuned on an instruction following dataset, and was used in relevant previous work.

The model was configured with the **temperature of 0.35** for balance between reproducibility and answer diversity (see **Appendix** for formula with explanation) and a **maximum of 1024 tokens** generated at each call.

Throughout this work, two other much smaller language models are used for various auxiliary tasks, introduced below:

Link Predictor

For prediction-based graph traversal, a relatively smaller LLM is used to predict edges. A smaller model is significantly faster at generation and can potentially be used in parallel with the main one to generate the processing queue ahead of time. Following the survey [42] and empirical tests (see **Appendix** for prompts, **Prediction-based**

traversal for method) on several candidate models, the best performing Qwen2.5-0.5B-Instruct[43] was selected. Qwen2.5-0.5B is a 500 million parameter (1GB) general purpose, linguistically diverse model that has a fine-tuned version for following instructions. The model was configured with the **temperature of 0.15** favouring reproducibility and a **400 generated tokens maximum**.

Semantic Encoder

In certain sections, embeddings and semantic search were used in retrieval and matching on the graph. Since the task is not to decode a sequence of text, but to efficiently encode the meaning of a sentence, a much smaller model sufficed. S-BERT[3] provides a library and a list of compatible models, along with their average performance. MPNet-base-v2 [44] is the best performing model across 20 benchmark datasets for semantic search and embedding. No additional setup was required for this model.

3.1.3 Accuracy Specification

All used QA datasets have unstructured free-format text gold and generated answers. To evaluate the correctness of such answers, 2 metrics are commonly used:

1. **Exact Match** (EM) also known as hits@1, provides a clear cut metric measuring the complete token match between the ground truth and the generated answer.
2. **F1-score** measures the overlap between the correct and the predicted answers. Computes a harmonic mean of precision and recall at the token level, allowing partial credit for the correct but incomplete or complete with extra text answers. $\beta = 1$ equally favours precision and recall, resulting in a balanced metric.

$$F_{\beta=1} = (1+\beta^2)\cdot\frac{\text{Precision}\cdot\text{Recall}}{(\beta^2\cdot\text{Precision})+\text{Recall}} = \frac{2\cdot\text{Precision}\cdot\text{Recall}}{\text{Precision}+\text{Recall}}$$

$$\text{Precision} = \frac{\#\ \text{Overlapping tokens}}{\#\ \text{Predicted tokens}}$$

$$\text{Recall} = \frac{\#\ \text{Overlapping tokens}}{\#\ \text{Ground truth tokens}}$$

In both cases, the ground truth and the predicted answers are preprocessed through lowercasing, removal of punctuation and word stemming. This is to prevent erroneous penalties for capital casing, hyphenation, punctuation and plural/singular word forms.

3.2 Baseline Performance

To establish the expected lower and upper boundaries on the accuracy of predicted results, we ran 2 types of tests on all datasets: closed-book and perfect-path tests, giving the expected minimum and the expected maximum performance respectively. Given the nondeterministic nature of probabilistic models, such as LLMs, even when provided

[3]`https://sbert.net/`

with the perfect context, they sometimes fail to produce the correct answer. For consistency of performance, both tests were ran twice and the results were averaged. Doing so for subsequent tests was not possible due to computational and time constraints[4].

3.2.1 Closed-Book Performance

Evaluates the model's intrinsic ability to answer questions without any external source of data. In this setup, the model relies exclusively on the knowledge it acquired in the pre-training stage and may reflect on its reasoning capabilities. The task is defined as

$$\hat{A} = \text{LLM}(Q)$$

3.2.2 Perfect-Path Performance

All datasets are constructed such that each question has at least one topic entity that can be matched on the graph. In the same way, every answer has at least one associated entity. The perfect path between them is considered to be the shortest possible in absence of actual reasoning paths.

This approach evaluates the performance given all shortest possible paths from each topic entity to all possible answer entities. Unlike the heuristic approaches generating noisy paths along the graph and potentially confusing the model, this approach ensures the perfect relevance of the context provided. It isolates the model's ability to interpret context from retrieving relevant data. The task is defined as

$$\hat{A} = \text{LLM}(Q, C^*)$$

$$C^* = \bigcup_{e_Q \times e_A} \arg\min_{P' \subseteq P} \left| P'(e_Q, e_A) \right|$$

where

C^*	: Perfect context containing all shortest paths between topic and answer entities
$e_{Q/A}$	: Set of question topic and answer entities (starting and destination points)
P	: Set of all possible paths between 2 nodes in a knowledge graph
$\lvert\cdot\rvert$	: Length of any given path
$\bigcup_E$	: Concatenation over all combinations of relevant entities $E = e_1 \times e_2$

[4]Unlike baseline prompts, populated with context on the fly, retrieval-based techniques require intermediate processing and ranking, and need significantly more time to answer a single question.

3.3 Relevance-Based Graph Pruning

Providing the entire graph to the model is not feasible due to the size of a graph and the context size limit of the LLM itself. This prompts for an algorithmic heuristic to narrow down the context to provide. The most obvious one is to define a relevance scoring function that determines how useful an entity, edge or a triplet is to answering the question, then sort the intermediate results by relevance and truncate the rest.

This also helps us test the model's ability to leverage the context it is provided with. Given the goal of developing an autonomous traversal agent, we experiment with completeness of the context at $\{n-1,\, n,\, n+1\}$ fixed-depth subgraphs. Such test will determine how robust the system is to imperfect context retrieval.

3.3.1 Task Formulation

We first experiment with scoring functions on fixed-depth subgraphs, and then introduce a way of dynamically traversing the graph, giving the LLM a choice to either keep exploring or stop and produce the final answer. The former task is defined as:

$$\hat{A} = \mathrm{LLM}(Q, C)$$

$$C = \textsc{GenerateTripletsContext}(Q, G, \mathbf{depth}, \mathbf{N})$$

The logic of how the context is constructed is described by Algorithms 1 and 2 respectively. We first identify the topic entities (provided in datasets); starting from each one, we then continuously populate a subgraph using a fixed-depth breadth-first search on the original graph that simply collects all nodes and edges.

We have 2 parameters **depth** and **N**, that define the subgraph depth or "hop-number" this project experiments with, and the maximum number of triplets we provide to the LLM respectively. N was limited to 250 empirically through numerous trials to achieve maximum GPU utilization and not run out of memory. Algorithm 1 also has a SCORE() function, which is where the heuristic to determine relevance shall be used.

Algorithm 1 Context from re-ranked fix-depth subgraph

```
function GenerateTripletsContext(question, KG, depth, N)
    $topic\_entities \leftarrow \textsc{ExtractEntities}(question)$
    $triplets \leftarrow \textsc{ExtractSubgraphTriplets}(entities, KG, \mathbf{depth})$
    $ranked\_triplets \leftarrow \mathbf{SCORE}(triplets, question)$
    $top\_triplets \leftarrow \mathrm{SORT}(ranked\_triplets)[:\mathbf{N}]$
    $context \leftarrow \textsc{CombineTriplets}(top\_triplets)$
      return $context$
end function
```

Algorithm 2 Subgraph triplets extractor

```
function ExtractSubgraphTriplets(entities, graph, depth)
    $subgraph\_entities \leftarrow \varnothing$
    for entity in entities do
        $node \leftarrow graph.\textsc{NodeFromEntity}(entity)$
        $bfs\_nodes \leftarrow \textsc{BreadthFirstSearch}(graph, node, \textbf{depth})$
        $subgraph\_entities \leftarrow subgraph\_entities \cup bfs\_nodes$
    end for
    $subgraph \leftarrow graph.\textsc{Subgraph}(subgraph\_entities)$
    $triplets \leftarrow []$
    for triplet in subgraph.edges do
        $triplets.\textsc{append}(triplet)$
    end for
    return triplets
end function
```

3.3.2 BM25

Okapi BM25 (Best Matching) [45] is a ranking function used extensively in Information Retrieval to estimate the relevance of a set of documents to a given search query. It is a Bag-of-Words (BoW) approach to ranking documents based on query terms appearing in each document, regardless of their proximity within the document. Since the document in this context is a relatively short knowledge graph triplet, the structure and order of the words are can be ignored as that would rapidly increase the complexity of the function and lack data for proper modelling. The scoring function is defined as follows

$$SCORE_{\mathrm{BM25}}(t,q) = \sum_{w \in q} \mathrm{IDF}(w) \cdot \frac{f(w,t) \cdot (k_1 + 1)}{f(w,t) + k_1 \cdot (1 - b + b \cdot \frac{|t|}{avgTl})},$$

where t is a triplet, q is the query, w is a query token, $f(w,t_i)$ is term frequency, $\mathrm{IDF}(w)$ is inverse document frequency, $avgTl$ is the average triplet length, and k_1, b are hyperparameters[5]. They are set to $k_1 = 1.5$ and $b = 0.75$ respectively as recommended by many IR books and practitioners to strike a balance between document length and term frequency. Selection of these values is a highly subjective process and requires empirical fine-tuning depending on the data and the needs of the retrieval system.

While the fraction component of the BM25 expression computes a score based on the token counts inside a document. The Inverse Document Frequency (IDF) provides a relative score based on the total number of documents this token appears in.

$$IDF(w) = \log\left(\frac{N - n(w) + 0.5}{n(w) + 0.5} + 1\right),$$

where N is the total number of documents, and $n(w)$ is the number of documents containing w. Combined together, they create a balanced metric that allows to evaluate how relevant a triplet is to a question with respect to a collection of triplets.

We use the Lucene BM25s [46] variant of the algorithm as a Python library [47], because it provides the most efficient implementation and has a well-documented API.

3.3.3 S-BERT

BM25 is a lexical retrieval method, meaning it is limited to the exact matching when it computes word frequencies. Even after preprocessing, there's a high chance that text in any arbitrary knowledge base will not exactly match and may be a synonym or have a typo. It also does not capture any abstract meaning of a sentence, in which words are used and can therefore confuse words with their homonyms (same form, different meaning), further reducing the relevance of a retrieved context.

An alternative scoring function is one that considers the semantic representation of both a question and a triplet. The rise of distributional semantic representations through deep learning has changed how semantic representation is created. Unlike early symbolic

[5] k_1 represents term-frequency saturation, i.e. how much it favours frequency. b is length normalization coefficient, i.e. how much it favoures short documents.

Algorithm 3 BM25 Triplet Ranker

```
function RANKTRIPLETSBM25(question, triplets)
    $q\_tokens \leftarrow$ TOKENIZE($question$)
    $triplets\_tokens \leftarrow triplets$.FOREACH(TOKENIZE)
    $retriever \leftarrow$ CREATEBM25INDEX($triplets\_tokens$)    ▷ Computes IDF
    $ranked\_triplets \leftarrow$ MINHEAP()
    for t_tokens in triplets_tokens do
        $ranked\_triplet \leftarrow retriever$.SCORE($t\_tokens, q\_tokens$)    ▷ $SCORE_{BM25}$
        $ranked\_triplets$.INSERT($ranked\_triplet$)
    end for
     return $ranked\_triplets$
end function
```

and rule-based approaches, novel neural techniques are capable of operating on out-of-vocabulary tokens and capture long-range dependencies by default. We use MPNet [44], described above, as an encoder to obtain a semantic representation of a query and a triplet. The scoring function is defined as:

$$SCORE_{\text{SBERT}} = \text{sim}(h(q), h(t_i))$$
$$sim(A,B) = \frac{A \cdot B}{|A| \cdot |B|}$$

where $h(x) \in \mathbb{R}^d$ is a dense embedding of a text mapped to a vector space ($d = 768$), and $\text{sim}(\cdot,\cdot)$ is a Euclidean (Cosine) distance similarity function. Since all similarities are now calculated independent of one another, they need to be manually sorted before selection. To efficiently select the top-N most relevant triplets, we use a **min-heap** data structure during the triplet pass, which provides both memory and computationally efficient insertions.

Algorithm 4 S-BERT Triplet Ranker

```
function RANKTRIPLETSSBERT($question, triplets$)
    $q\_embedding \leftarrow$ SBERT.EMBED($question$)
    $triplet\_min\_heap \leftarrow$ MINHEAP()
    for triplet in triplets do
        $t\_embedding \leftarrow$ SBERT.EMBED($triplet$)
        $score \leftarrow$ COSINESIMILARITY($q\_embedding, t\_embedding$)
        $triplet\_min\_heap$.INSERT($score, triplet$))
    end for
     return $triplet\_min\_heap$
end function
```

3.4 Prediction-Based Graph Traversal

The main limitation of the pruning approaches described in the sections above is that they are fixed-depth and provide no way of dynamically traversing the graph for context retrieval. Moreover, they immediately fetch an entire subgraph, rather than construct it incrementally.

To address this, we develop a graph traversal algorithm that incrementally constructs the context using a Breadth-First Exploration with Beam-Search and Path Tracking. It concurrently explores several paths along the graph until it reaches maximum depth or decides that it has enough context to answer the question.

Inspired by Think-on-Graph [48], which leverages LLM as a reasoning agent to determine relevance and sufficiency of the context, we introduce Think-on-Graph with Link Prediction (ToG-LP), which replaces some calls to LLM with a blend of heuristics that significantly reduce processing time.

Think-on-Graph

The original implementation of the algorithm keeps track of a beam with top-N paths and has 3 main steps at each iteration D:

1. **Relation exploration** first retrieves all candidate in- and out-going relations linked to the tail entities of paths from the previous iteration p^{D-1}. These relations are then scored by relevance using the LLM and a specialized prompt, marking the first efficiency bottleneck.
2. **Entity exploration** then selects all entities from the triplets formed by the initial entities and the candidate relations, and performs a similar relevance scoring with LLM. This step also involves a costly call to the main model.
3. **Reasoning**. Once the candidate paths are extended, they are combined and the LLM is once again prompted, only this time to decide whether or not it has enough context to answer the original question.

Finally, when the exploration and reasoning phase is concluded, one last call is made to obtain the final answer returned for evaluation. In total, this results in $(2N+1)\cdot D+1$ calls to generate an answer. Upon the initial tests with this approach, it appeared to be very slow,[6] rendering it unfeasible to run a full test on all datasets.

We retain the reasoning and the final answer generation calls to the main model, but replace the relation and entity filtering components with more efficient heuristics: Link prediction and tail pruning respectively. This reduces the total number of calls to $D+1$.

The heuristic is chosen to combine explainability of rule-based and the flexibility of neural approaches in context retrieval logic.

[6]Over 10 minutes for a single question.

3.4.1 Link Prediction and Retrieval

To overcome the limitations of BM25 and S-BERT retrieval, and get around expensive calls to a capable reasoning model, we combine them and achieve the best of both worlds. A smaller, linguistically diverse LLM is prompted with the initial list of entities e^{D-1} and a formatted example[7] to produce **N** candidates. The output is parsed to extract the predicted relevant relations, and then, using the semantic encoder model each prediction is matched with the most similar existing relation available at this step. The scores are retained for further use. Algorithm 5 describes the prediction and matching logic as pseudocode.

Algorithm 5 Link Prediction Algorithm

```
function PREDICTNEDGES($question, entities, N$)
    $predicted\_edges \leftarrow$ LINKPREDICTORLLM($question, entities, N$)
    $candidates \leftarrow$ EXTRACTEDGES($predicted\_edges$)
    $relations \leftarrow graph.$RELSFROMENTITIES($entities$)
    $scored\_edges \leftarrow$ MINHEAP()
    for candidate in candidates do
        $best\_match \leftarrow$ SBERT.BESTMATCH($candidate, relations$)
        $scored\_edges.$INSERT($best\_match$)                ▷ contains the score
    end for
     return $scored\_edges$
end function
```

3.4.2 Tail pruning

Since edge selection is only one component of the context selection, to complete the triplet, we also need the tail entity for the iteration to complete. Certain edges are more prominent and may, therefore have several candidate entities associated with it. Consider an example with the states of the USA, the entity ”UnitedStatesOfAmerica” would have 50 entities connected to it through the ”StateOf” edge for each of the 50 states it has. Thus, if we predicted the outgoing link ”StateOf” as the one to explore, we would have at least 50 candidate entities to select from.

Again, this can be formalized as the problem of deciding how relevant something is to answering the question. To define a relevance scoring function, we need a set of comparable characteristics. Those can be obtained using either primary or secondary features of nodes and edges. Primary features include the data and metadata for each item, while secondary ones are representative of the item’s position in the graph and might include degree, relative connectivity, clustering coefficient, and others.

For example, we could use Node2Vec [50] to create a distributed representation similar to S-BERT [51]. However, that would be computationally expensive and would lack incremental learning if we were to continuously populate the graph. Alternatively,

[7]The model is provided with a worked example and is asked to justify each awarded score. Chain-of-Thought[49] prompting explores this tendency and shows that when asked for detailed answers, LLMs tend to perform and adhere to the format better.

BetaE [52] embeddings encode the nodes as Beta distributions and allow to implicitly model logical operations as latent space transformations. While more amenable to scaling and incremental updates, the method still requires substantial computational overhead.

In this work, we focus on a simpler ad-hoc approach that can be computed on the fly with the sole goal of speeding up answer generation. We assume that the entities connected to the existing path are more important and relevant than others and compute a **local connectivity score**.

For each candidate triplet, we combine the previously computed edge score (predicted by smaller LLM) and the local connectivity score with respect to the already constructed paths along the graph. The top-N triplets are then used to extend the paths, which are combined into the context and checked for sufficiency.

figure with a visual of beam search on triplets

Given a relation and a set of candidate entities, for each triplet in every step of the path ref figure, we check whether the graph contains an edge between each candidate and the head or tail of each triplet. For each existing connection, a score of 0.5 is awarded, and the score is first normalised by the step size (width of a step) and at the end by the path size (number of steps). This is done to make paths comparable and prevent the explosion of the score for numeric stability. Pseudocode describing the logic can be found below in Algorithm 6.

Algorithm 6 Tail selection. Local connectivity ranking

```
function LocalConnectivityRank($paths, entity, edge, candidate\_tails$)
    $scored\_triplets \leftarrow$ MinHeap()
    for tail in candidate_tails do
        $tail\_score \leftarrow 0$
        for step in paths do
            $step\_score \leftarrow 0$
            for triplet in step do
                $head\_score \leftarrow 0.5$ if $graph$.HasEdge($tripletHead, tail$) else 0
                $tail\_score \leftarrow 0.5$ if $graph$.HasEdge($tripletTail, tail$) else 0
                $step\_score$ += $head\_score + tail\_score$
            end for
            $tail\_score$ += $\frac{step\_score}{length(step)}$                ▷ Normalize over step
        end for
        $final\_score = edge.score * \frac{tail\_score}{length(paths)}$                ▷ Normalize over path
        $triplet \leftarrow (entity, edge, tail)$
        $scored\_triplets$.Insert($final\_score, triplet$)
    end for
     return $scored\_triplets$
end function
```

3.4.3 Reasoning

Once the paths list is extended with a set of new triplets, at the end of each iteration we prompt the main model to decide if it can answer the question or if it needs more information. This is perhaps the most important step of the algorithm, as we entirely rely on the model's ability to correctly interpret the question and the context without an explicit decision-making objective set before. This is possible due to zero-shot learning emerging with larger model sizes. As described in the background, the performance seems to be quite robust allowing researchers to develop more flexible algorithms. With all components combined together, the final ToG-LP traversal algorithm is described in Algorithm 7.

Algorithm 7 Think-on-Graph-Link Prediction Traversal Algorithm

```
function ToG-LP-Answer($question, topicEntities, maxDepth, beamWidth$)
    $exploredEntities = \text{SET}(topicEntities)$
    $paths = [$
      $[(entity,)$ for entity in topicEntities$]$
    $]$                                            ▷ 0. Initial paths with single entities
    for depth = 1 to maxDepth do
        $tailEntities \leftarrow \text{ExtractTails}(paths[depth-1])$      ▷ tails from $p^{D-1}$
        $unexplored \leftarrow tailEntities - exploredEntities$
        $exploredEntities \leftarrow exploredEntities \bigcup unexplored$
        $edgeHeap \leftarrow \text{MinHeap}()$                  ▷ 1. Predict relations
        for entity in tailEntities do
            $predictedEdges \leftarrow \textbf{PredictEdges}(question, entity)$
            $edgeHeap.\text{Insert}(predictedEdges)$           ▷ contains entity
        end for
        $topEdges \leftarrow edgeHeap[: beamWidth]$
        $tripletHeap \leftarrow \text{MinHeap}()$                ▷ 2. Select tails
        for (scoredEdge, entity) in topEdges do
            $tails \leftarrow graph.\text{GetTails}(entity, scoredEdge)$
            $rankedTriplets = \textbf{LocalConnectivityRank}(paths, entity, scoredEdge, tails)$
            $tripletHeap.\text{Insert}(rankedTriplets)$
        end for
        $topTriplets \leftarrow tripletHeap[: beamWidth]$
        $paths.\text{Extend}(topTriplets)$
        if EnoughContext(question, paths) then              ▷ 3. Reason
            break
        end if
    end for
     return GenerateAnswer($question, paths$)             ▷ 4. Generate Answer
end function
```

Chapter 4

Results

In this chapter, we present the results of experiments for the 3 datasets in the 4 setups described above. We first establish the baseline performance on the selected model to obtain the expected upper and lower boundaries for further experiments and compare them with the existing State-of-the-Art (SOTA). We then evaluate 2 different triplet ranking methods: BM25 and S-BERT respectively, to determine how under, over and perfect context retrieval affect the model's ability to answer a question. We then study the performance of the Think-on-Graph Link Prediction algorithm, comparing it to the two previous techniques. And finally, we present the performance on different types of questions provided by the ComplexWebQuestions dataset.

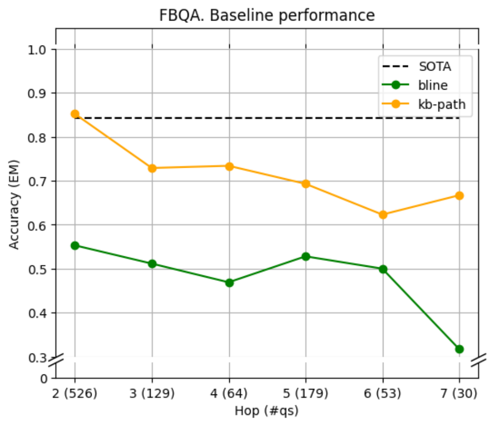
4.1 Baseline and Previous Studies

Given the nondeterministic nature of generative models, even when provided with the perfect context to answer the question, it is not guaranteed that the model will produce the correct answer. To get an estimate for the likely performance we run closed-book and perfect-path tests, respectively. In the former case, the model has to answer the question with its internal knowledge it acquired during pre-training. In the latter, we provide the joint set of all shortest possible paths between every combination of topic (identified in a question) and answer entities, which is treated as perfect path context.

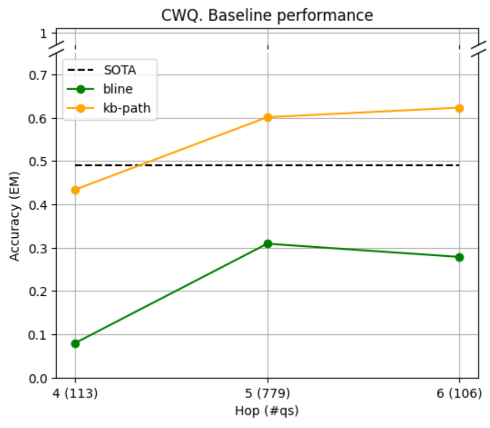
The results of the two runs for each dataset are displayed in Figure 4.1 as green for baseline, yellow for perfect-path respectively, and the horizontal dotted line for average SOTA performance identified for each dataset. Graphs display Exact Match (EM) accuracy against the number of hops needed to answer a question annotated with the number of questions in that category.

Given the rapidly improving SOTA with more research appearing weekly, the cutoff date was chosen to be October 2024. SOTA methods ([53] for FBQA, [54] for CWQ and for MetaQA respectively) integrate external knowledge into the QA process, either by generating SPARQL queries or by re-ranking triplets on a portion of the graph[1].

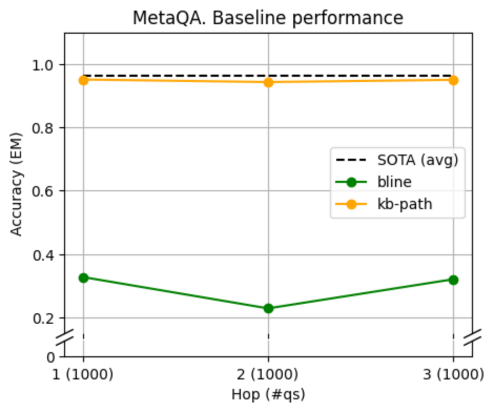
[1]It is important to note that the SOTA results were obtained on OpenAI GPT4-Turbo, which is a significantly more powerful model (approximately 220 times larger).



(a) FBQA



(b) CWQ



(c) MetaQA

Figure 4.1: Baseline performance graphs. The yellow line presents perfect-path performance with the known path along the graph. The green line presents baseline performance purely on training data. Dashed line presents the average SOTA performance on each dataset as of October 2024.

Method	FBQA	CWQ	MetaQA	Average
SOTA	0.842	0.491	0.963	0.765
Baseline	0.480	0.222	0.291	0.331
Perfect-Path	0.716	0.553	0.948	0.739
Increase	0.236	0.331	0.657	**0.408**

Table 4.1: Average baseline performance. Each row represents individual performance for each dataset on a fixed-depth with the average in the final column. Final row presents perfect-path performance increase over baseline for each dataset with the average in the final column.

FreebaseQA (FBQA) is a trivia-style set of questions about general knowledge that varies in depth of retrieval, built from the Freebase knowledge graph. ComplexWebQuestions (CWQ) is a more advanced Freebase-based set of questions that provides

4 types of question requiring more complex context comprehension, question types include: composition, conjunction, comparison, and superlative identification. MetaQA is a movie-specific fact retrieval dataset with question varying from 1 to 3 hops on the WikiMovies knowledge graph.

The baseline performance, when the model is only provided with the question, produces an average accuracy of 33% reflecting on model's ability to retrieve knowledge acquired in pre-training stage.

For all datasets, providing correctly retrieved relevant knowledge gives an average 41% improvement in accuracy. For MetaQA and ComplexWebQuestions (CWQ), the accuracy reaches or slightly surpasses average SOTA, while FBQA remains 15-20% below steadily decreasing with the number of hops needed to answer the question. All further experiments are expected to produce performance that lies in this range. The average perfect-path accuracy is only 2.5% below that of SOTA showing competitive results and the potential to use Mistral7B [41] as a QA agent.

MetaQA gains the highest increase of 66% when the model is presented with the context containing correct answer. FBQA and CWQ get 24% and 33% improvement respectively. CWQ, even with the SOTA method only reaches about 49% average accuracy.

Although it is not directly appropriate to compare the performance with the number of hops across all datasets together, there does seem to be a trend on hops 4 through 6 with a slight increase followed by a plateau.

4.2 Relevance-Based Methods

As a preparatory step for the dynamic traversal algorithm, we would like to understand how does under, over and correct retrieval of the context affect the model's ability to answer a question. Since the ultimate goal of the system is to be an autonomous agent, it is reasonable to expect occasional under or over-completeness of the provided context, making it more difficult to extract together the final answer.

To test robustness of the model towards completeness of the context, we assume the agent is moving in the correct direction but would like to study 3 edge cases, where it almost reaches $(n-1)$, reaches (n) and goes further than it should $(n+1)$ on the graph to retrieve the relevant knowledge. The case curves on the graph are coloured as follows: blue for $n-1$, red for n and purple for $n+1$.

BM25

BM25 is a ranking function based on term counts in a collection of documents (graph triplets). For each question, we identify the hop number, which indicates how far apart the topic and answer entities are on the knowledge graph, and use it to define case depth (± 1). We then retrieve the subgraph of a given depth and apply the ranking function with respect to the question. The function is used it to re-rank subgraph triplets and the best scoring 250[2] triplets are provided as context to the model.

The results in Figure 4.2 and Table 4.2 show that the performance on all datasets lies between the two boundaries we established earlier. Average accuracy varies between 40% and 47% depending on completeness of the context, with the perfect context retrieval being 4% worse than $n-1$.

On FBQA the accuracy is 18% above the baseline and 6% below the perfect at the average of 66%, seemingly following the shape of the perfect-path, indicating the model is reaching some kind of intrinsic limit. Over the 3 cases, the results only differ by about 1% on average.

CWQ gets a marginal 0.7% bump with perfect context and above. Relative to the baseline, CWQ does not get any improvement with BM25 context retrieval and sits at 21% accuracy. The average deviation in results is 0.4%.

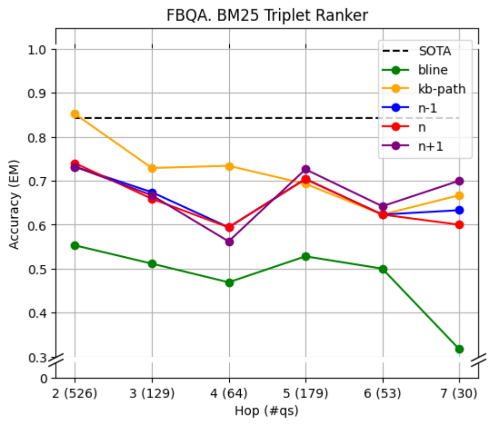
For MetaQA, ranking on an extra depth level shows a maximum 22% improvement, although still 44% below the perfect. Unlike FBQA and CWQ, a steady increasing trend is observed from 33% to 51% depending on context completeness, amounting to the standard deviation of 10%

Overall, the 3 cases show gradually improving accuracy with $n+1$ being better on average. Context retrieval with BM25 causes an average standard deviation of 4%, most of which comes from MetaQA. Small overall deviation for FBQA, CWQ and an increasing trend for MetaQA show an increase in resulting answer accuracy.

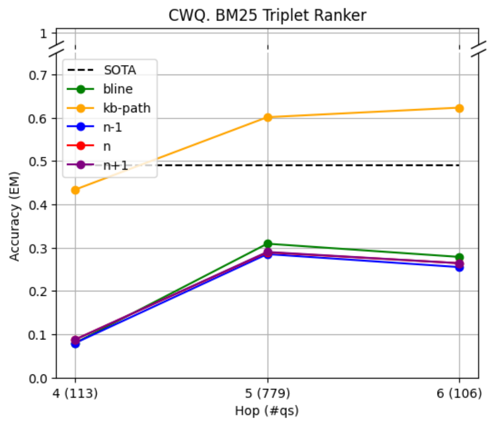
Method	FBQA	CWQ	MetaQA	Average
$n-1$	0.660	0.207	0.331	0.399
n	0.653	0.214	0.485	0.442
$n+1$	0.672	0.214	0.511	**0.466**
Std. Deviation	0.010	0.004	0.097	0.037

Table 4.2: Average BM25 triplet ranker performance. Each row represents individual performance for each dataset on a fixed-depth with the average in the final column. Final row presents standard deviation of each dataset with the average in the final column.

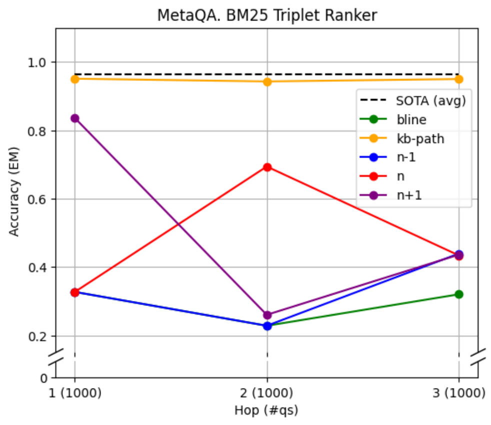
[2]Empirically selected for maximum utilisation while staying within GPU memory capacity.



(a) FBQA



(b) CWQ



(c) MetaQA

Figure 4.2: BM25 triplet ranker performance. Green line for baseline, yellow for perfect-path, representing lower and upper expected boundaries. Blue lines for $n-1$ depth (under-retrieval), red for n (perfect retrieval), purple for $n+1$ (over-retrieval). Dashed line shows average SOTA for each dataset as of October 2024.

S-BERT

Sentence-BERT is a small language model trained specifically to encode semantic meaning of word sentences into a latent vector representation. We use this model to compare the similarity of a question with each triplet to decide which ones are more relevant and sort them based on Euclidian vector distance. In the same way, the top triplets of a fixed-depth subgraph retrieved for each question are later provided to the LLM along with the question to produce the final answer.

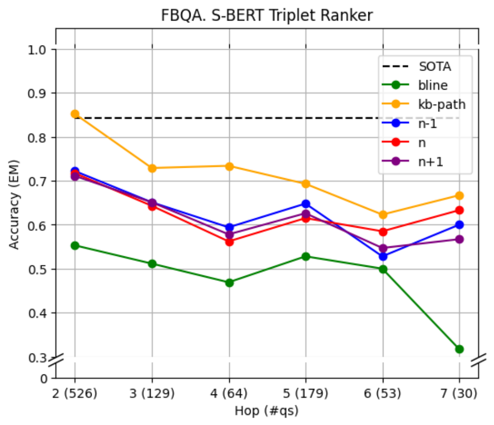
The results are shown in Figure 4.3 and Table 4.3. As before, performance lies between the two boundaries, rendering them a useful metric for evaluating knowledge retrieval heuristics. FBQA sits at about 62% average accuracy 4% below the BM25 performance and 10% below the perfect. CWQ gets an 11% improvement over BM25 bringing it closer, but still not reaching the 55% upper boundary. All cases for both FBQA and CWQ follow the perfect line with no significant discrepancy between the 3 settings throughout the hops.

MetaQA gets a 13% increase in maximum accuracy from BM25 and a clearer distinction of 16% between n and $n+1$, whereas $n-1$ remains at the baseline level improving slightly for 3 hop questions. It is important to note that $n+1$ produces high accuracy on 1-hop questions, but lower than n for 2-hop questions with everything else the same.

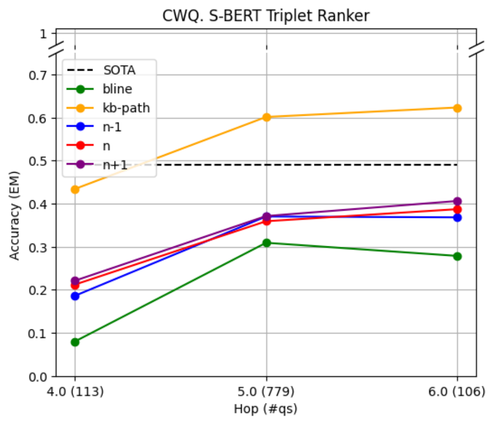
Overall, S-BERT produces a 2.3% higher standard deviation in accuracy of under- and over-retrieval performance, and has an average 7% increase in accuracy compared to BM25. This shows that S-BERT retrieval is more sensitive to depth setting, but produces better results. $n+1$ is also the best settings in this case, allowing us to conclude that retrieving more context is a desirable outcome provided that the ranking function can determine the relevance well and makes S-BERT the best ranker of the two methods.

Method	FBQA	CWQ	MetaQA	Average
$n-1$	0.624	0.308	0.330	0.421
n	0.626	0.319	0.483	0.476
$n+1$	0.613	0.333	0.647	**0.531**
Std. Deviation	0.007	0.013	0.159	0.060

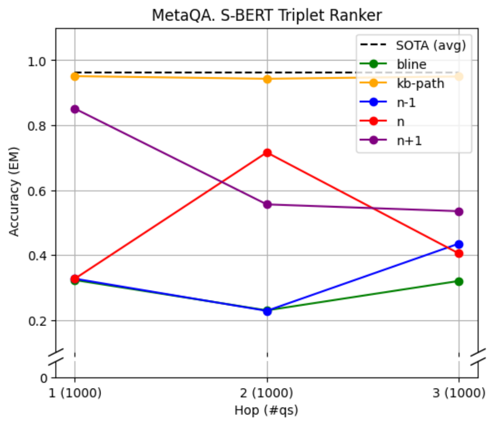
Table 4.3: Average S-BERT triplet ranker performance. Each row represents individual performance for each dataset on a fixed-depth with the average in the final column. Final row presents standard deviation of each dataset with the average in the final column.



(a) FBQA



(b) CWQ



(c) MetaQA

Figure 4.3: S-BERT triplet ranker performance. Green line for baseline, yellow for perfect-path, representing lower and upper expected boundaries. Blue lines for $n-1$ depth (under-retrieval), red for n (perfect retrieval), purple for $n+1$ (over-retrieval). Dashed line shows average SOTA for each dataset as of October 2024.

4.3 ToG-LP

Think-on-Graph Link Prediction is a graph traversal algorithm that iteratively constructs a beam of several most likely candidate paths along the graph. At each iteration we first use a smaller LLM to predict 5 candidate relations to explore from the previous tail nodes, the 5 candidates are then matched with the most likely real edges using S-BERT. Then, given the head nodes and scored edges, each potential triplet gets a score combined of the edge and each candidate tail node's local connectivity score that is computed based on its connections to the already constructed path. At the end of each iteration, the main LLM is prompted with the currently available context to decide if it's enough to answer the question or if it should continue exploring until reaching the maximum depth.

While our modification is focused on improving the time requirements, the final duration is largely driven by the maximum depth hyperparameter. Replaced main LLM calls still require time and at the time we were only able to run the algorithm on a single setting of depth to 4. This was an arbitrarily chosen value that balances the depth and the complexity of questions available in the datasets and at the same time was computationally feasible in time given.

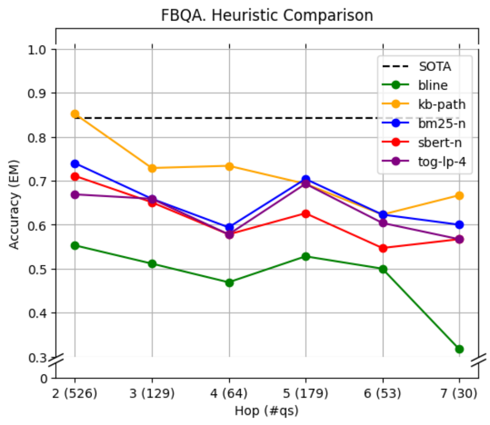
For FBQA, ToG-LP-4 outperforms the baseline by about 15%. CWQ, on the other hand performs 6% worse than the baseline. MetaQA follows with 2.6% reduction in accuracy compared to baseline.

Relative to relevance-based methods, ToG-LP outperforms S-BERT by about 1.5% but underperforms BM25 by 4% on the FBQA dataset. On CWQ, ToG is 16% worse than S-BERT and 5% worse than BM25 triplet rankers. MetaQA has the highest accuracy difference of -22% compared to S-BERT and -23% with BM25. It should be noted that while on FBQA ToG-LP-4 produces competitive results, on CWQ and MetaQA they are suboptimal even for baseline.

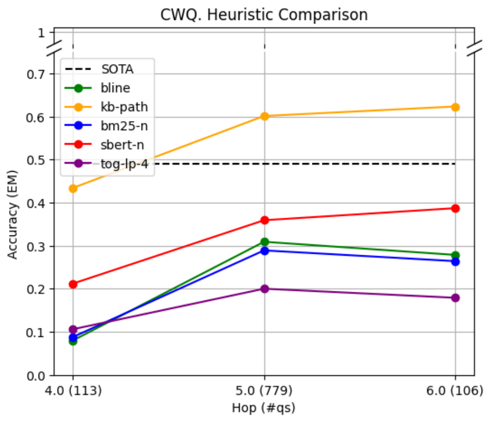
On average, ToG-LP-4 has an accuracy of 35%, which is just 2% above the baseline and 39% below the perfect-path. It underperforms both BM25 and S-BERT by 10% and 12% respectively. The overall performance of ToG-LP-4 is suboptimal, but contributes to research towards autonomous QA agents and requires further in-depth analysis and experimentation.

Method	FBQA	CWQ	MetaQA	Average
BM25-*n*	0.653	0.214	0.485	0.451
S-BERT-*n*	0.613	0.319	0.483	**0.472**
ToG-LP-4	0.628	0.162	0.265	0.352
Increase over S-BERT	0.015	-0.157	-0.218	-0.120

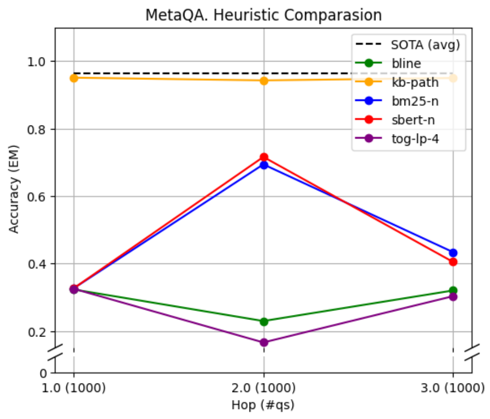
Table 4.4: Average *n* performance compared with ToG-LP. Rows represent performance of each heuristic on each dataset with the average in the final column. Final row represents increase in accuracy of ToG-LP above S-BERT with the average in the final column.



(a) FBQA



(b) CWQ



(c) MetaQA

Figure 4.4: Knowledge retrieval heuristics performance comparison. Blue lines represent BM25 with perfect (n) context retrieval depth. Red lines represent S-BERT with perfect context retrieval depth performance. Purple lines represent ToG-LP performance with the maximum depth of 4.

Chapter 5

Discussion

In this chapter, we examine the final experimentation results in more detail hypothesising about the origin of trends, factors affecting the performance, implications and limitations of the work. We also do a case study of the system's performance on different types of questions as provided by the ComplexWebQuestions dataset.

5.1 Baseline

We established lower and upper boundaries of the expected performance on Mistral7B on each of the datasets. On average, the upper boundary, where the model is presented with the perfect attainable path along the knowledge graph, is 41% better than without it. This significant increase justifies the cause for using LLMs augmented with external knowledge in automated QA. Across the 3 datasets varying in tasks and retrieval complexity, the improvement is very clear and consistent.

FBQA and CWQ get a 24% and 33% increase from 48% and 22%, respectively, whereas MetaQA gets mean 66% improvement from 29% baseline. This difference in the initial and improved performance can likely be attributed to the selection of training corpus used by Mistral7B [41] in the pre-training stage. While Mistral does not publicly disclose the training set, one could suppose that given the data requirements for obtaining emergent capabilities [17], it might contain parts of the web in it, usually sourced from the CommonCrawl.

Since Freebase is largely composed of data available on the public web until 2015, there is a possibility of bias towards that data causing any Freebase-based question set to have a higher baseline performance. One way to address this is to run the experiments on a model with known training data to eliminate bias.

Average perfect-path performance on CWQ is 55%, whereas average SOTA performance is 49%, which is signifying that the dataset is still challenging to models, such as Mistral7B and GPT4-Turbo [54]. Even when provided with relevant context, almost half the time, the models fails to produce the correct answer, which might be a manifestation of the limitations of emergent reasoning capabilities in LLMs and prompts for further research.

5.2 Relevance-Based Methods

We employ 2 heuristics that introduce the concept of relevance scoring between the query and triplets on the knowledge graph. Experiments with lexcial BM25 and semantic S-BERT are designed to test the model's ability to interpret the provided context in 3 scenarios. The scenarios are: under-retrieval $(n-1)$ where the retrier lacks 1 step from reaching the answer, perfect (n), and over-retrieval $(n+1)$ where the context contains more information than it should.

These cases show deviations in accuracy based on completeness of the context, which are useful for building autonomous QA agents. When the agent traverses the graph, it is likely to end up in the vicinity of the answer, rather than on the actual node [48]. The experiments help understand the degree to which the 3 edge case scenarios affect the final accuracy.

BM25

FBQA and MetaQA get a 15-20% increase, whereas CWQ goes below the baseline. For CWQ, neither case yields any improvement, with all results sat just below the baseline, possibly showing that either the context retrieved is incorrect, or the model is incapable of processing the context. As described above (Chapter 3, ComplexWebQuestions), FBQA contains basic fact retrieval questions, while CWQ needs more complex comprehension, such as comparative or superlative reasoning.

MetaQA quite sensitive to context retrieval depth, and exhibits an expected increasing trend from $n-1$ to $n+1$. One reason this is the case could be the afforementioned Freebase bias and the intricate balance between context knowledge and the model's internal knowledge. Given low baseline performance and an assumption that little WikiMovies data ended up in Mistral's training set, we could see a clearer performance distinction between the different scenarios. One way to study this further is to pick a model with a publicly available training data, which does not have Freebase/WikiMovies data in it eliminating the potential bias.

BM25 produces an average accuracy of 44% with 3.7% standard deviation, which is an 11% increase from the average baseline level. Low standard deviation shows a relatively stable response to the 3 scenarios, which is desired in an autonomous system. Between the 3 settings, there is little deviation, allowing us to infer 2 main conclusions.

First, BM25 hyperparameters[1] k_1 and b, might need fine-tuning. This is especially evident on MetaQA, where $n+1$ performs almost 3% better than n, showing that BM25 works better on a large collection of documents. Second, the knowledge graph data retrieved could be only marginally relevant with respect to model's internal knowledge in QKV matrices (introduced in Chapter 2, self-attention), which take precedence when generating the answer. Both theories require further experiments with more data collected and more detailed analysis.

[1]Favouring term frequency and document length respectively.

S-BERT

Instead of using exact lexical matches, S-BERT embeds sentences as dense vectors in a latent space that results in similar words appearing together [51]. Synonymous words are better identified mathematically, without relying on exact word matches. On average, S-BERT based graph pruning produces a 48% accuracy with a 6% standard deviation. This is higher than BM25, but more sensitive to the depth setting, which is expected.

CWQ gets a more notable increase and a clearly improving accuracy with the extra depth of the context provided, almost 11% compared with BM25 and 13% with baseline. FBQA, on the other hand, performs better without additional context by about 1%. MetaQA continues having an increasing trend, from the same $n-1$ accuracy, to a 13% higher mean, which drives the standard deviation to increase to 16%.

Overall, S-BERT has the best average accuracy across all datasets at 53% with 6% standard deviation, making it a more sensitive, but better ranking method for constructing the context relevant to answering domain-specific questions.

5.3 ToG-LP

Think-on-Graph Link Prediction is a graph traversal algorithm that iteratively constructs several paths along the graph in a beam search manner and lets the main LLM model decide if it needs to continue traversing at the end of each iteration. Inspired by the approach, the original ToG [48] paper introduces, we modify the algorithm replacing costly LLM calls with fast ad-hoc scoring and pruning heuristics. We use a smaller LLM to predict the 5 candidate edges to explore and match each one of the most similar existing edge at that step using S-BERT's semantic encoding capabilities. We For each of the candidate tail nodes, we then comput a local connectivity score with the respect to the existing path and combine the two, selecting the best ones to extend the paths beam.

Due to computational and time costs of running the algorithm, even though it is faster the original implementation, we still only had enough time to run experiments for a single configuration with the maximum depth of 4. The results were presented in the Figure 4.4 and Table 4.4.

The average accuracy was 35%, which is 2% above the baseline, but is significantly below relevance scoring peers, 10% and 12% below BM25 and S-BERT respectively. ToG-LP managed to produce competitive results on FBQA but went below the baseline level for CWQ and MetaQA. The likely cause for that is bias towards Freebase and has to be explored further.

Due to how the testing data was collected, it was not possible to study the performance of individual components of the algorithm in detail. Future work should address this by collecting and analysing more fine-grained data on the algorithm.

As the 2 main components driving the exploration are both heuristics, it is worth pointing out the limitations of either, which significantly contribute to the final performance.

First, a small LLM, while offering faster inference, has a reduced context window size becoming a bottleneck further along the graph, and much weaker ”emergent capabilities” [17], which are the bedrock of neural QA paradigm.

Local connectivity score is perhaps the most limiting factor, while it is logically sound and quick to compute, it suffers the drawbacks of statistical approach, mainly data sparsity. Because the graphs are incomplete or sparsely connected, and paths are relatively short, the score would be biased towards hub nodes. Graph structure needs to be studied in more detail to design a more appropriate tail pruning heuristic.

5.4 Case Study: CWQ Question Types

ComplexWebQuestions [38] constructs a dataset that requires complex reasoning along the Freebase [35] knowledge graph. The dataset contains 4 types of questions (as shown in Table 5.2).

Question Type	**Example**
Comparative	What country with an ISO less than 233 borders Russia?
Composition	What actress plays Claire on a television show with the theme song ’Lonely Girl’?
Conjunction	In what country is Arabic spoken and is the birthplace of Barbara Starr?
Superlative	From all the sights in Madrid, what is the latest exhibition venue that opened in that city?

Table 5.1: Different types of questions and their examples as provided by the ComplexWebQuestions dataset.

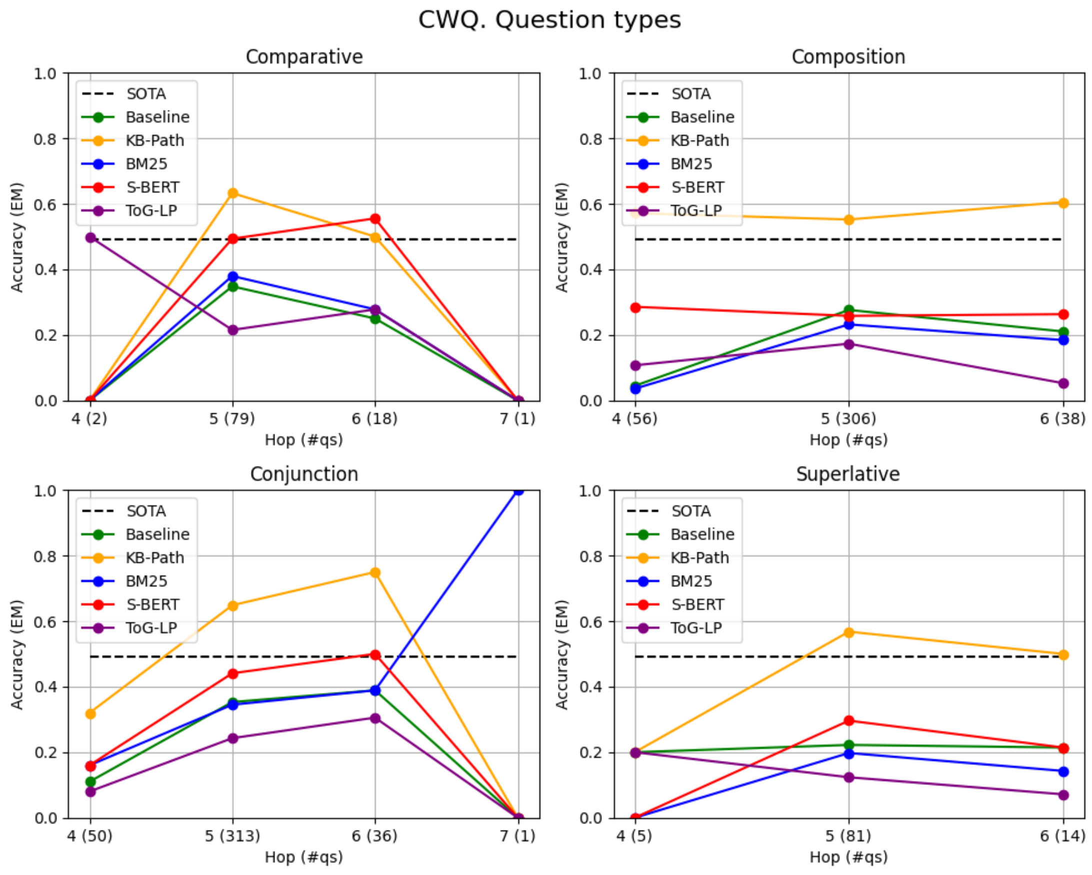


Figure 5.1: Mistral7B performance on CWQ question types. Coloured lines present different heuristics, dashed line presents the current SOTA.

Method	Comparative	Composition	Conjunction	Superlative	Average
Baseline	0.150	0.177	0.213	0.212	0.188
Perfect-Path	0.283	**0.576**	0.430	0.423	0.428
BM25-*n*	0.164	0.151	0.473	0.113	0.225
S-BERT-*n*	0.262	0.269	0.275	0.170	**0.244**
ToG-LP-4	0.248	0.111	0.157	0.132	0.162

Table 5.2: Average performance for each type of CWQ question. Rows represent different heuristics and their performance on each type of question, the final column is the average.

include the table with averages and comment on what methods performs best for each question type. Comment on class balance when selecting a subset

Chapter 6

Conclusions

In this chapter, we summarise key findings and draw conclusions from them. We compare what was achieved against what the original objectives were, what limitations and the implications are, and highlight areas for researched further that have the potential for improving automated QA with LLMs.

6.1 Summary

In this work, we have explored the field of automated QA and the modern neural techniques to perform it, selected models, data and defined some heuristics to explore. We have developed a framework for automated testing and evaluation. We conducted extensive experiments and laid out objectives for future work.

I have:

- Understood the landscape
- Selected appropriate data
- Selected appropriate models
- Defined a task
- Built a framework for loading data, standardizing, running (plug n play, parallel) and evaluating.
- Comprehensively experimented with heuristics and parameters
- Analysed the results and identified limitations
- Laid out objectives for future works

The results demonstrate a clear relationship between retrieval quality and model performance, with the baseline accuracy significantly lower than the perfect-path accuracy across all datasets. The 41% average improvement when moving from baseline to perfect-path highlights the importance of providing the model with correct and complete contextual information. This is particularly evident in MetaQA, where the baseline is the lowest, suggesting that the model lacks internal knowledge to answer those questions without external data. However, even with perfect context, the model does not always reach SOTA performance, reinforcing the idea that retrieval alone is not enough—how the model integrates and reasons over the retrieved knowledge also plays a crucial role.

Between the two triplet-ranking approaches, S-BERT consistently outperforms BM25, with an average 7% accuracy gain. This is expected, as BM25 relies on term frequency and does not capture semantic meaning, whereas S-BERT is optimized for sentence similarity, making it particularly effective in datasets like CWQ, where questions involve comparative reasoning and more abstract relationships. Interestingly, both methods show that retrieving slightly more context (n+1) is better than retrieving too little (n-1), suggesting that models can handle some degree of noise in the retrieved data as long as relevant information is included. However, the gains from retrieval expansion are not always linear, and in some cases, over-retrieval (n+1) does not significantly improve performance compared to retrieving exactly the needed information (n).

ToG-LP, on the other hand, underperforms compared to both BM25 and S-BERT, showing an average 12% drop compared to S-BERT. The performance loss is particularly striking in MetaQA (-21% compared to S-BERT), suggesting that ToG-LP struggles with effective knowledge selection in multi-hop reasoning tasks. One possible reason for this underperformance is that graph-based link prediction may not be as effective as direct triplet ranking when applied to the QA task, particularly when the structure of the graph does not align well with how the model processes retrieved information. Additionally, ToG-LP's performance in CWQ is notably weak (worse than BM25 and significantly below S-BERT), indicating that the approach does not generalize well to datasets requiring more nuanced reasoning.

Despite its lower accuracy, ToG-LP's method of dynamically predicting missing links still holds potential. It might be beneficial in hybrid approaches, where ToG-LP is used to augment existing triplet rankings rather than replace them. One key observation is that ToG-LP does not always retrieve the most relevant context efficiently, which might explain why it falls short compared to direct ranking-based methods. Future improvements could focus on refining how it selects relevant paths, possibly integrating a confidence score to filter out less relevant predicted links. Another area for improvement could be adjusting its depth traversal strategy, ensuring it does not generate too many loosely connected or irrelevant paths.

Overall, these results highlight that triplet ranking is currently a more reliable method than link prediction for knowledge retrieval in QA. S-BERT emerges as the best approach, particularly in complex question scenarios, while BM25 remains a simple but relatively effective baseline. ToG-LP, while promising, requires further refinement before it can compete with these methods. Its potential role may not be as a standalone ranking approach but rather as a complementary technique to enhance retrieval, particularly in cases where explicit knowledge connections are missing.

The highly relevant, precise context massively increases accuracy. Irrelevant/confusing context can improve performance but only marginally, it still tends to follow the baseline curve

6.2 Challenges and Limitations

- Only have a subset of the graphs
- Limited/biased data (freebase, wikimovies)
- Only compared exact matches (EM) in results
- The relevance-based methods use a prefetched subgraph, instead of constructing it incrementally
- Only considered +-1 on retrieval evaluation
- Only have 1 try at ToG
- Local Connectivity Score counts both head and tail and only considers the paths up to now
- Querying graphs (semantic + text) is a bottleneck - slow to load and process

6.3 Future Work

- Collect better data (more specific, better questions)
- More research into graph structure (small world networks, clustering coefficients)
- Experiment with BM25 hyperparameters
- Vary deltas 2,3,4, etc for n+-Δ to study the effect.
- Look at retrieval results in more detail (hits@)
- Run on models of different sizes (2B, 7B, 32B, 72B)
- Run on models with open training data (to account for bias)
- Run BM25 and S-BERT incrementally, rather than on the fix-depth portion
- Experiment with other tail selection methods (node2vec)
- Explore faster graph querying technique
- Redo ToG beam search with nodes rather than triplets (makes more sense)
- Timing ToG

Bibliography

[1] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, USA, 3rd edition, 2009. ISBN 0136042597.

[2] Ali Mohamed Nabil Allam and Mohamed Hassan Haggag. The question answering systems: A survey. 2012.

[3] Ellen Riloff and Michael Thelen. A rule-based question answering system for reading comprehension tests. In *Proceedings of the 2000 ANLP/NAACL Workshop on Reading Comprehension Tests as Evaluation for Computer-Based Language Understanding Sytems - Volume 6*, ANLP/NAACL-ReadingComp '00, page 13–19, USA, 2000. Association for Computational Linguistics. doi: 10.3115/1117595.1117598. URL `https://doi.org/10.3115/1117595.1117598`.

[4] Bernhard Waltl, Georg Bonczek, and Florian Matthes. Rule-based information extraction: Advantages, limitations, and perspectives. 2018. URL `https://api.semanticscholar.org/CorpusID:216006964`.

[5] Hongshen Chen, Xiaorui Liu, Dawei Yin, and Jiliang Tang. A survey on dialogue systems: Recent advances and new frontiers. *ACM SIGKDD Explorations Newsletter*, 19(2):25–35, November 2017. ISSN 1931-0153. doi: 10.1145/3166054.3166058. URL `http://dx.doi.org/10.1145/3166054.3166058`.

[6] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013. URL `https://arxiv.org/abs/1301.3781`.

[7] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017. URL `http://arxiv.org/abs/1706.03762`.

[8] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks, 2014. URL `https://arxiv.org/abs/1409.3215`.

[9] Alexander Clark, Gianluca Giorgolo, and Shalom Lappin. Statistical representation of grammaticality judgements: the limits of n-gram models. In Vera Demberg and Roger Levy, editors, *Proceedings of the Fourth Annual Workshop on Cognitive Modeling and Computational Linguistics (CMCL)*, pages 28–36, Sofia, Bulgaria, August 2013. Association for Computational Linguistics. URL `https://aclanthology.org/W13-2604/`.

[10] Justin T. Chiu and Alexander M. Rush. Scaling hidden markov language models, 2020. URL `https://arxiv.org/abs/2011.04640`.

[11] Hakime Öztürk, Arzucan Özgür, Philippe Schwaller, Teodoro Laino, and Elif Ozkirimli. Exploring chemical space using natural language processing methodologies for drug discovery. *Drug Discovery Today*, 25, 02 2020. doi: 10.1016/j.drudis.2020.01.020.

[12] Mark Johnson. How the statistical revolution changes (computational) linguistics. In Timothy Baldwin and Valia Kordoni, editors, *Proceedings of the EACL 2009 Workshop on the Interaction between Linguistics and Computational Linguistics: Virtuous, Vicious or Vacuous?*, pages 3–11, Athens, Greece, March 2009. Association for Computational Linguistics. URL `https://aclanthology.org/W09-0103/`.

[13] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. *J. Mach. Learn. Res.*, 3(null):1137–1155, March 2003. ISSN 1532-4435.

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

[15] Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer, 2020. URL `https://arxiv.org/abs/2004.05150`.

[16] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Peiyu Liu, Jian-Yun Nie, and Ji-Rong Wen. A survey of large language models, 2025. URL `https://arxiv.org/abs/2303.18223`.

[17] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed H. Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, Jeff Dean, and William Fedus. Emergent abilities of large language models, 2022. URL `https://arxiv.org/abs/2206.07682`.

[18] 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, 2020. URL `https://arxiv.org/abs/2005.14165`.

[19] Shervin Minaee, Tomas Mikolov, Narjes Nikzad, Meysam Chenaghlu, Richard Socher, Xavier Amatriain, and Jianfeng Gao. Large language models: A survey, 2025. URL `https://arxiv.org/abs/2402.06196`.

[20] Linmei Hu, Zeyi Liu, Ziwang Zhao, Lei Hou, Liqiang Nie, and Juanzi Li. A

survey of knowledge enhanced pre-trained language models, 2023. URL `https://arxiv.org/abs/2211.05994`.

[21] Xiaokai Wei, Shen Wang, Dejiao Zhang, Parminder Bhatia, and Andrew Arnold. Knowledge enhanced pretrained language models: A compreshensive survey, 2021. URL `https://arxiv.org/abs/2110.08455`.

[22] Neel Nanda, Lawrence Chan, Tom Lieberum, Jess Smith, and Jacob Steinhardt. Progress measures for grokking via mechanistic interpretability. *ArXiv*, abs/2301.05217, 2023. URL `https://api.semanticscholar.org/CorpusID:255749430`.

[23] Zorik Gekhman, Gal Yona, Roee Aharoni, Matan Eyal, Amir Feder, Roi Reichart, and Jonathan Herzig. Does fine-tuning llms on new knowledge encourage hallucinations?, 2024. URL `https://arxiv.org/abs/2405.05904`.

[24] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33:9459–9474, 2020.

[25] Yongfeng Zhang, Qingyao Ai, Xu Chen, and Pengfei Wang. Learning over knowledge-base embeddings for recommendation. 03 2018. doi: 10.48550/arXiv.1803.06540.

[26] Bhaskarjit Sarmah, Benika Hall, Rohan Rao, Sunil Patel, Stefano Pasquali, and Dhagash Mehta. Hybridrag: Integrating knowledge graphs and vector retrieval augmented generation for efficient information extraction, 2024. URL `https://arxiv.org/abs/2408.04948`.

[27] Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, and Ting Liu. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *ACM Transactions on Information Systems*, 43(2):1–55, January 2025. ISSN 1558-2868. doi: 10.1145/3703155. URL `http://dx.doi.org/10.1145/3703155`.

[28] Qinggang Zhang, Shengyuan Chen, Yuanchen Bei, Zheng Yuan, Huachi Zhou, Zijin Hong, Junnan Dong, Hao Chen, Yi Chang, and Xiao Huang. A survey of graph retrieval-augmented generation for customized large language models, 2025. URL `https://arxiv.org/abs/2501.13958`.

[29] Pengcheng Qiu, Chaoyi Wu, Xiaoman Zhang, Weixiong Lin, Haicheng Wang, Ya Zhang, Yanfeng Wang, and Weidi Xie. Towards building multilingual language model for medicine, 2024.

[30] Ilias Chalkidis, Manos Fergadiotis, Nikolaos Manginas, Eva Katakalou, and Prodromos Malakasiotis. Regulatory compliance through Doc2Doc information retrieval: A case study in EU/UK legislation where text similarity has limitations. In Paola Merlo, Jorg Tiedemann, and Reut Tsarfaty, editors, *Proceedings of the 16th Conference of the European Chapter of the Association for Computational*

Linguistics: Main Volume, pages 3498–3511, Online, April 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.eacl-main.305. URL `https://aclanthology.org/2021.eacl-main.305/`.

[31] Ankit Pal, Logesh Kumar Umapathi, and Malaikannan Sankarasubbu. Medmcqa: A large-scale multi-subject multi-choice dataset for medical domain question answering. In Gerardo Flores, George H Chen, Tom Pollard, Joyce C Ho, and Tristan Naumann, editors, *Proceedings of the Conference on Health, Inference, and Learning*, volume 174 of *Proceedings of Machine Learning Research*, pages 248–260. PMLR, 07–08 Apr 2022. URL `https://proceedings.mlr.press/v174/pal22a.html`.

[32] Di Jin, Eileen Pan, Nassim Oufattole, Wei-Hung Weng, Hanyi Fang, and Peter Szolovits. What disease does this patient have? a large-scale open domain question answering dataset from medical exams, 2020. URL `https://arxiv.org/abs/2009.13081`.

[33] Shuangjia Zheng, Jiahua Rao, Ying Song, Jixian Zhang, Xianglu Xiao, Evandro Fei Fang, Yuedong Yang, and Zhangming Niu. Pharmkg: a dedicated knowledge graph benchmark for bomedical data mining. *Briefings in Bioinformatics*, 22(4):bbaa344, 12 2020. ISSN 1477-4054. doi: 10.1093/bib/bbaa344. URL `https://doi.org/10.1093/bib/bbaa344`.

[34] Zahra Sepasdar, Sushant Gautam, Cise Midoglu, Michael A. Riegler, and Pål Halvorsen. Enhancing structured-data retrieval with graphrag: Soccer data case study, 2024. URL `https://arxiv.org/abs/2409.17580`.

[35] 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*, SIGMOD '08, page 1247–1250, New York, NY, USA, 2008. Association for Computing Machinery. ISBN 9781605581026. doi: 10.1145/1376616.1376746. URL `https://doi.org/10.1145/1376616.1376746`.

[36] Google. Freebase data dumps. `https://developers.google.com/freebase/data`, ¡year¿.

[37] Kelvin Jiang, Dekun Wu, and Hui Jiang. FreebaseQA: A new factoid QA data set matching trivia-style question-answer pairs with Freebase. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 318–323, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1028. URL `https://aclanthology.org/N19-1028/`.

[38] Alon Talmor and Jonathan Berant. The web as a knowledge-base for answering complex questions, 2018. URL `https://arxiv.org/abs/1803.06643`.

[39] Alexander Miller, Adam Fisch, Jesse Dodge, Amir-Hossein Karimi, Antoine Bordes, and Jason Weston. Key-value memory networks for directly reading documents, 2016. URL `https://arxiv.org/abs/1606.03126`.

[40] Yuyu Zhang, Hanjun Dai, Zornitsa Kozareva, Alexander J Smola, and Le Song. Variational reasoning for question answering with knowledge graph. In *AAAI*, 2018.

[41] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7b, 2023. URL `https://arxiv.org/abs/2310.06825`.

[42] Fali Wang, Zhiwei Zhang, Xianren Zhang, Zongyu Wu, Tzuhao Mo, Qiuhao Lu, Wanjing Wang, Rui Li, Junjie Xu, Xianfeng Tang, Qi He, Yao Ma, Ming Huang, and Suhang Wang. A comprehensive survey of small language models in the era of large language models: Techniques, enhancements, applications, collaboration with llms, and trustworthiness, 2024. URL `https://arxiv.org/abs/2411.03350`.

[43] Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report, 2025. URL `https://arxiv.org/abs/2412.15115`.

[44] Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. Mpnet: Masked and permuted pre-training for language understanding, 2020. URL `https://arxiv.org/abs/2004.09297`.

[45] K. Sparck Jones, S. Walker, and S.E. Robertson. A probabilistic model of information retrieval: development and comparative experiments: Part 1. *Information Processing Management*, 36(6):779–808, 2000. ISSN 0306-4573. doi: https://doi.org/10.1016/S0306-4573(00)00015-7. URL `https://www.sciencedirect.com/science/article/pii/S0306457300000157`.

[46] Chris Kamphuis, Arjen P. de Vries, Leonid Boytsov, and Jimmy Lin. Which bm25 do you mean? a large-scale reproducibility study of scoring variants. In Joemon M. Jose, Emine Yilmaz, João Magalhães, Pablo Castells, Nicola Ferro, Mário J. Silva, and Flávio Martins, editors, *Advances in Information Retrieval*, pages 28–34, Cham, 2020. Springer International Publishing. ISBN 978-3-030-45442-5.

[47] Xing Han Lù. Bm25s: Orders of magnitude faster lexical search via eager sparse scoring, 2024. URL `https://arxiv.org/abs/2407.03618`.

[48] Jiashuo Sun, Chengjin Xu, Lumingyuan Tang, Saizhuo Wang, Chen Lin, Yeyun Gong, Lionel M. Ni, Heung-Yeung Shum, and Jian Guo. Think-on-graph: Deep and responsible reasoning of large language model on knowledge graph, 2024. URL `https://arxiv.org/abs/2307.07697`.

[49] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2023. URL `https://arxiv.org/abs/2201.11903`.

[50] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks, 2016. URL `https://arxiv.org/abs/1607.00653`.

[51] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 11 2019. URL `https://arxiv.org/abs/1908.10084`.

[52] Hongyu Ren and Jure Leskovec. Beta embeddings for multi-hop logical reasoning in knowledge graphs, 2020. URL `https://arxiv.org/abs/2010.11465`.

[53] Kexuan Sun, Nicolaas Paul Jedema, Karishma Sharma, Ruben Janssen, Jay Pujara, Pedro Szekely, and Alessandro Moschitti. Efficient and accurate contextual re-ranking for knowledge graph question answering. In Nicoletta Calzolari, Min-Yen Kan, Veronique Hoste, Alessandro Lenci, Sakriani Sakti, and Nianwen Xue, editors, *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pages 5585–5595, Torino, Italia, May 2024. ELRA and ICCL. URL `https://aclanthology.org/2024.lrec-main.496/`.

[54] Guanming Xiong, Junwei Bao, and Wen Zhao. Interactive-kbqa: Multi-turn interactions for knowledge base question answering with large language models, 2025. URL `https://arxiv.org/abs/2402.15131`.

Appendix A

Supplementary material

A.1 Temperature

The probability of selecting a token w_i given a temperature T is:

$$P(w_i) = \frac{\exp\left(\frac{s_i}{T}\right)}{\sum_j \exp\left(\frac{s_j}{T}\right)}$$

where:

- s_i is the logit (raw score) for token w_i.
- T is the temperature.

High temperature ($T > 1$): Increases randomness by making the probability distribution of the tokens more uniform. Encourages more creative selection in a beam-search decoding.

Low temperature ($T < 1$): Makes the output more deterministic by amplifying the differences between the logits (raw scores). Less creative and more predictable.

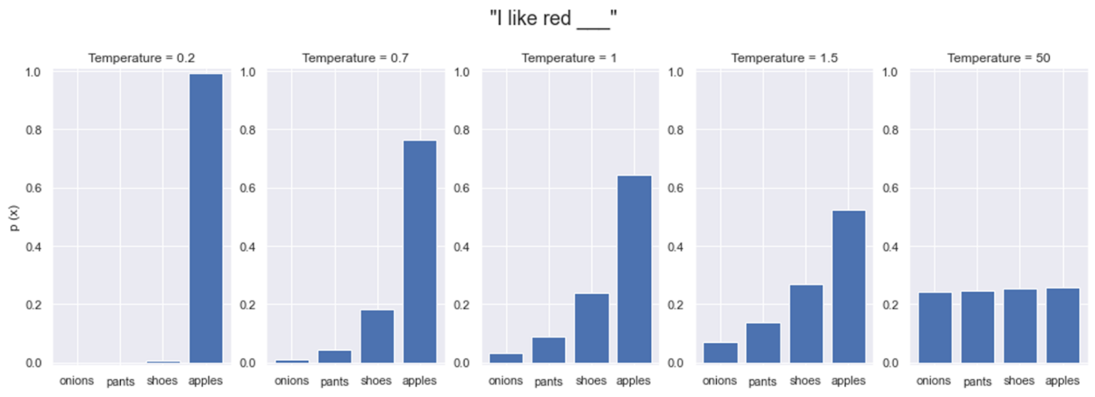


Figure A.1: Temperature setting and the final distribution.
Taken from `https://www.hopsworks.ai/dictionary/llm-temperature`

A.2 Dataset verification

A.3 Prompts

Appendix B

Results Tables

Due to how much space the final result tables take up, elaborate versions of all results were moved to the appendix and are listed below.

B.1 FBQA

Table B.1: FBQA results. Question Hops 2,3,4

	hops	2		3		4	
	Metric	EM	F1	EM	F1	EM	F1
Method	Hops						
bline	0	0.553	0.649	0.519	0.610	0.469	0.586
bline2	0	0.553	0.649	0.504	0.598	0.469	0.587
kb	1	0.732	0.771	0.643	0.650	0.594	0.650
	2	0.740	0.778	0.674	0.670	0.547	0.639
	3	0.732	0.768	0.659	0.661	0.594	0.665
	4	0.000	0.000	0.667	0.671	0.594	0.665
	5	0.000	0.000	0.000	0.000	0.562	0.636
	6	0.000	0.000	0.000	0.000	0.000	0.000
	7	0.000	0.000	0.000	0.000	0.000	0.000
	8	0.000	0.000	0.000	0.000	0.000	0.000
	9	0.000	0.000	0.000	0.000	0.000	0.000
	10	0.000	0.000	0.000	0.000	0.000	0.000
	11	0.000	0.000	0.000	0.000	0.000	0.000
kb-path	0	0.852	0.885	0.729	0.741	0.734	0.753
sbert	1	0.722	0.733	0.643	0.650	0.594	0.645
	2	0.717	0.692	0.651	0.658	0.547	0.604
	3	0.711	0.695	0.643	0.635	0.594	0.639
	4	0.000	0.000	0.651	0.640	0.562	0.583
	5	0.000	0.000	0.000	0.000	0.578	0.615
	6	0.000	0.000	0.000	0.000	0.000	0.000
	7	0.000	0.000	0.000	0.000	0.000	0.000
	8	0.000	0.000	0.000	0.000	0.000	0.000
	9	0.000	0.000	0.000	0.000	0.000	0.000
	10	0.000	0.000	0.000	0.000	0.000	0.000
	11	0.000	0.000	0.000	0.000	0.000	0.000
tog-lp-4	4	0.669	0.351	0.659	0.332	0.578	0.322

Table B.2: FBQA results. Question Hops 5,6,7

	hops		5		6		7
	Metric	EM	F1	EM	F1	EM	F1
Method	Hops						
bline	0	0.520	0.631	0.491	0.524	0.267	0.475
bline2	0	0.536	0.630	0.509	0.541	0.367	0.525
kb	1	0.721	0.749	0.660	0.685	0.600	0.646
	2	0.687	0.731	0.642	0.657	0.633	0.675
	3	0.693	0.735	0.585	0.637	0.633	0.699
	4	0.704	0.740	0.000	0.000	0.000	0.000
	5	0.704	0.744	0.623	0.660	0.000	0.000
	6	0.726	0.756	0.623	0.663	0.633	0.696
	7	0.000	0.000	0.642	0.654	0.600	0.682
	8	0.000	0.000	0.000	0.000	0.700	0.785
	9	0.000	0.000	0.000	0.000	0.000	0.000
	10	0.000	0.000	0.000	0.000	0.000	0.000
	11	0.000	0.000	0.000	0.000	0.000	0.000
kb-path	0	0.693	0.719	0.623	0.660	0.667	0.706
sbert	1	0.615	0.638	0.604	0.611	0.633	0.688
	2	0.648	0.668	0.509	0.538	0.700	0.730
	3	0.620	0.637	0.528	0.548	0.700	0.731
	4	0.648	0.675	0.000	0.000	0.000	0.000
	5	0.615	0.641	0.528	0.573	0.000	0.000
	6	0.626	0.650	0.585	0.583	0.600	0.650
	7	0.000	0.000	0.547	0.576	0.633	0.683
	8	0.000	0.000	0.000	0.000	0.567	0.625
	9	0.000	0.000	0.000	0.000	0.000	0.000
	10	0.000	0.000	0.000	0.000	0.000	0.000
	11	0.000	0.000	0.000	0.000	0.000	0.000
tog-lp-4	4	0.693	0.368	0.604	0.320	0.567	0.313

Table B.3: FBQA results. Question Hops 8,9,10

	hops	8		9		10	
	Metric	EM	F1	EM	F1	EM	F1
Method	Hops						
bline	0	0.333	0.463	0.889	0.889	1.000	0.833
bline2	0	0.333	0.460	0.889	0.889	1.000	0.833
kb	1	0.833	0.750	0.778	0.867	1.000	1.000
	2	0.667	0.538	0.778	0.867	1.000	0.828
	3	0.833	0.765	0.778	0.867	0.750	0.766
	4	0.000	0.000	0.000	0.000	0.000	0.000
	5	0.000	0.000	0.000	0.000	0.000	0.000
	6	0.000	0.000	0.000	0.000	0.000	0.000
	7	0.667	0.556	0.000	0.000	0.000	0.000
	8	0.667	0.583	0.667	0.756	0.000	0.000
	9	0.500	0.389	0.778	0.867	0.750	0.766
	10	0.000	0.000	0.778	0.867	1.000	1.000
	11	0.000	0.000	0.000	0.000	1.000	1.000
kb-path	0	0.500	0.417	0.778	0.778	0.750	0.812
sbert	1	0.500	0.417	0.778	0.778	0.750	0.528
	2	0.667	0.643	0.889	0.889	0.500	0.600
	3	0.667	0.583	0.778	0.778	0.500	0.500
	4	0.000	0.000	0.000	0.000	0.000	0.000
	5	0.000	0.000	0.000	0.000	0.000	0.000
	6	0.000	0.000	0.000	0.000	0.000	0.000
	7	0.667	0.597	0.000	0.000	0.000	0.000
	8	0.667	0.583	0.889	0.889	0.000	0.000
	9	0.667	0.597	0.889	0.889	0.500	0.645
	10	0.000	0.000	0.889	0.889	0.500	0.634
	11	0.000	0.000	0.000	0.000	0.500	0.634
tog-lp-4	4	0.833	0.417	0.778	0.389	0.750	0.403