Studies on Question Answering in Open-Book and Closed-Book Settings

Tareq Alkhaldi

August 2023

Abstract

Natural language is the medium through which humans mainly communicate and store their knowledge to pass it on. Manual analysis and handling of such massive quantities is beyond human capabilities. Consequently, Natural Language Processing (NLP) has been an active and fruitful field, as automatically processing such information yields immense benefits to many applications such as machine translation, text summarization, and question answering (QA).

Searching for information is typically done by asking questions in search engines and going through the returned results to find a specific piece of information in mind, which can be tiresome and time-consuming. A trained QA model can automatically read those documents and return the exact answer immediately, making our lives easier. Furthermore, QA systems inherently reflect the reasoning capabilities and the level of various natural language understanding (NLU) tasks. Enhancing QA systems promotes the advancement of all these related tasks. Therefore, in this thesis, we focus on the QA task.

Based on complexity, the QA task is divided into two categories: single-hop QA, which requires a single reasoning step, and multi-hop QA, which necessitates more than one reasoning step. On the other hand, based on context utilization, the QA task is divided into open-book QA and closed-book QA. The former involves always providing the context documents of the question to the model, while the latter only inputs a question. Both scenarios have their strengths and weaknesses, and each can be suitable for certain use cases. For example, using context permits accurate answer extraction but is bottlenecked by the accuracy of document retrieval systems. On the other hand, a closed-book setting is not limited by the quality or speed of retrieval systems as it can memorize much

factual knowledge during pre-training and consequently is faster; however, it is vulnerable to hallucinations. Therefore, it is important to investigate and improve both scenarios.

This thesis studies both scenarios. First, we focus on the setting of multi-hop questions in open-book QA, as single-hop performance already surpasses human performance. Multi-hop models are required to identify and collect important pieces of information from the provided context documents to achieve the answer. Researchers have developed models that utilize the identified supporting sentences; however, we find that they do so ineffectively. We propose a method to focus on such information more effectively and find additional beneficial signals from the context documents.

Second, we investigate the setting of closed-book QA. Previous studies confirmed and measured the amount of factual knowledge that models memorize during pre-training, yet the mechanism and inner workings are not investigated. We address this shortcoming by investigating how models memorize factual knowledge, the responsible parts inside a model, if the knowledge is concentrated in certain places, and applying such information by experimenting with pruning less concentrated areas to reduce the model size without performance loss.

Finally, we study and improve multi-hop QA in a closed-book setting. Previous work has focused on single-hop questions for studying models in closed-book QA. To close the gap, we concentrate on multi-hop QA and its improvements via question decomposition. We decompose the complex question into sub-questions explicitly, or implicitly by the chain-of-thoughts of a model. We also detect promising tips for further improvement.

Acknowledgments

First and foremost, I want to express my gratitude and praise to God Almighty, who has bestowed me with countless blessings and paved the way for me to reach where I am today.

I have been fortunate to receive extensive support and assistance from many people throughout my years at Kyoto University, and I can't thank them enough.

I would like to extend my sincere gratitude and appreciation to my supervisor, Professor Sadao Kurohashi, for his precious guidance, constant support, and kindness, despite his busy schedule. He helped me recognize invaluable wholesome perspectives about my research and assisted me in steering my research on the right path. On a personal level, I greatly appreciate his constant regard for my and my family's well-being. His radiant and welcoming smile makes approaching and communicating with him easy. I consider it a privilege to graduate from his laboratory.

I would also like to thank the members of my thesis examination committee: Professor Tatsuya Kawahara and Professor Hisashi Kashima, for their valuable advice and feedback.

I am also grateful to Professor Daisuke Kawahara for his helpful suggestions and feedback. I also deeply thank Associate Professor Chenhui Chu for his significant contribution and constant help in revising and polishing my work. I am also thankful to Dr. Yugo Murawaki, Dr. Fei Cheng, Dr. Ribeka Tanaka, Dr. Tomohide Shibata, and Dr. Fabien Cromieres for sharing their expertise and great advice.

I am truly grateful to have had the opportunity to work alongside a remarkable group of labmates in the Kurohashi Laboratory. In particular, I would like to

express my appreciation to Dr. Raj Dabre, Dr. Tomohiro Sakaguchi, Dr. Shuhei Kurita, Dr. Arseny Yolmachev, Mr. Yudai Kishimoto, Dr. Yin-Jou Huang, Dr. Hirokazu Kiyomaru for all the enjoyable research discussions and research tips, and for their comments on my draft papers. I also thank Mr. Takashi Kodama, Mr. Song Haiyue, Ms. Qianying Liu, Mr. Naota Ueda, Mr. Kazumasa Omura, Mr. Zhuoyuan Mao, and Mr. Shuichiro Shimizu for their assistance, delightful conversations, and company.

I also thank our secretaries, Ms. Yuko Ashihara, Ms. Naho Yoshitoshi, and Ms. Terumi Kosugi, for assisting me with all the administrative work and procedures.

Last but not least, I deeply thank my dear parents, my lovely wife, and my kids for their support and encouragement to persist, and for their patience during my lengthy academic journey. I am grateful to all my family and friends for their understanding and support during times when we were unable to be present for important milestones and occasions back home.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Most of today's knowledge is stored in natural language using many mediums, like blogs, news articles, and books. Automatically processing these huge amounts of text provides significant advantages, as humans cannot manually analyze such quantities of text. Therefore, natural language processing (NLP) has been a hot topic recently.

Within the last decade, machines have become more and more powerful and accelerated. Not only allowing for faster analysis but also enabling newer model architectures. Specifically, neural network-based models have become feasible. While the predominant methods for NLP depended on the statistical analysis of the text, the new model architectures caused a start of a new era of neural NLP.

Early neural architectures mainly consisted of feed-forward networks (Bengio et al., 2000); however, such models had no notion of sequentiality. Newer architectures that focus on sequential input are recurrent neural networks (RNNs) (Mikolov et al., 2010), and its descendant, long-short term memory networks (LSTMs) (Graves, 2012). They permitted the training of an impactful model called sequence-to-sequence (Seq2Seq), which takes a sequence of items, like words or characters, and outputs another sequence. Seq2Seq is effective for machine translation and chatbot applications.

A limitation of RNNs is that they have to squeeze the features of a whole sen-

tence into one hidden state to represent it. Generating the output sequence could only access this hidden state to generate the output sentence. To address this, one of the most significant core innovations is the attention mechanism (Bahdanau et al., 2015), which allows the model to focus on other text parts selectively, not limiting the model to one hidden state. The Transformer (Vaswani et al., 2017) is a recent outstanding emergent architecture based on the attention mechanism. Typically, models like BERT (Devlin et al., 2019) and GPT-2 (Radford et al., 2018) are constructed by stacking Transformer blocks. While GPT-2 excels at Language Modeling, where it predicts the next word given the previous words, BERT is primarily known for Masked Language Modeling, a task where it predicts masked or hidden words within a given context. After the language models are trained on billions of words of natural text, they learn how to understand the natural text and produce output that often matches human-level performance. Then these foundation models are fine-tuned with labeled data for downstream tasks. Such a paradigm unlocked improvements in many applications, such as machine translation, text summarization, and question-answering (QA).

1.2 Question Answering

1.2.1 Importance of the QA Task

QA is a particularly interesting task as it is the main way people look for information. When using search engines, we're often looking for a certain piece of information, and we are faced with many matching web pages which we have to go through to find the desired answer. A trained QA model would read those documents automatically and return the exact answer directly, saving us time and effort.

Another practical case is to use it as part of a chatbot that extracts information from documents or articles of frequently asked questions (FAQs) for a certain organization. This keeps the knowledge management effort low while providing straightforward and natural access to the stored knowledge. QA systems can also help with the decision-making process by providing relevant information to support making a well-informed decision. This can be applicable in domains such

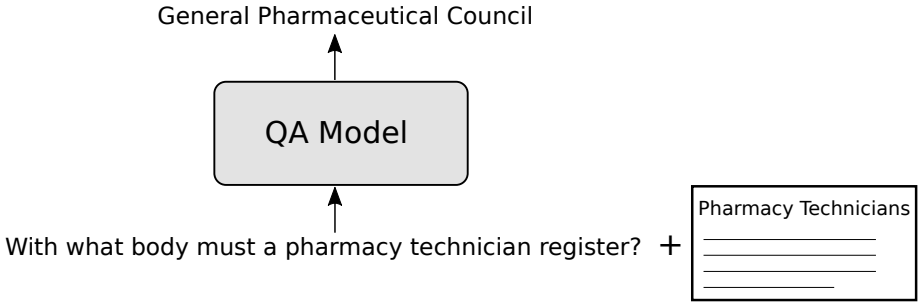

Figure 1.1: An example of single-hop QA.

as healthcare, finance, and the legal field.

QA systems require a deep understanding of context, making them a good testbed to evaluate natural language understanding (NLU). This deep understanding spans a lot of NLU tasks that a QA model needs to be skilled at, for example, coreference resolution, named entity recognition, textual entailment, etc. Therefore, promoting improvements in QA systems intrinsically advances these tasks.

The QA tasks can be divided by question complexity to single-hop QA and multi-hop QA and by the availability of external resources to open-book QA and closed-book QA. We explain each task in detail.

1.2.2 Single-hop QA

A straightforward way of using a QA model is to provide the context document alongside the question as input to the model and let it predict an answer. When questions are simple in nature, requiring a single document to extract the answer without the need for complex reasoning or multi-step inference, it is called single-hop QA. Nonetheless, even with simple questions, the models should be able to grasp subtle contextual nuances and handle ambiguities that may arise within the given text. Figure 1.1 shows an example of single-hop QA.

Earlier studies primarily focused on addressing simple questions. In spite of their relative simplicity, the remarkable improvements in the accuracy and effectiveness of the models have paved the way for practical applications in various domains.

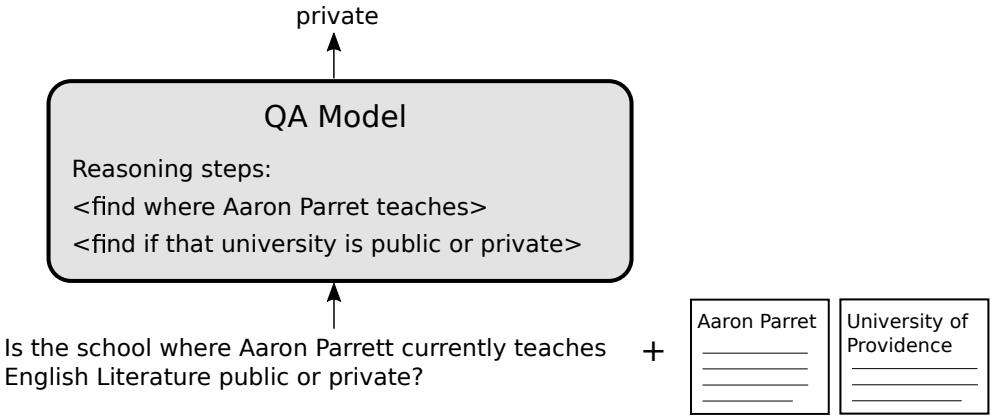

Figure 1.2: An example of multi-hop QA.

Models for reading comprehension of single-hop questions have learned to surpass human-level performance (Lan et al., 2020; Zhuang et al., 2021) on datasets like SQuAD (Rajpurkar et al., 2018), even without additional complex architecture other than the pre-trained Transformers. However, leading models for single-hop questions are found to lack the intended reasoning abilities and fail when tested on more complex questions requiring more than one reasoning step, called multi-hop questions (Yang et al., 2018).

1.2.3 Multi-hop QA

To explore the complexity of addressing complex questions, researchers have introduced several multi-hop QA datasets (Yang et al., 2018; Ho et al., 2020; Trivedi et al., 2022). One prominent example of such a dataset is called HotpotQA (Yang et al., 2018), which challenges models to tackle multi-hop questions that necessitate evidence gathering from two or more paragraphs to derive the final answer. Figure 1.2 illustrates an example of a multi-hop question.

Unlike single-hop questions that provoke the development of direct question-matching skills for extracting answers from a single context document, multi-hop questions demand more sophisticated reasoning abilities from the QA models. The models must not only identify relevant information from the initial question but also traverse multiple paragraphs to collect supporting evidence sentences

essential for arriving at the correct answer. This requirement fosters the development of advanced reasoning mechanisms, enabling models to perform complex inferences. Addressing multi-hop questions comes with its own set of challenges. The models must effectively deal with coreference resolution, entity linking, and context-dependent information extraction to accurately gather relevant evidence across paragraphs.

Furthermore, some datasets, like HotpotQA, encourage models to not merely predict the final answer but also identify and provide the sentences within the context that justify their predictions. This feature contributes to enhancing the transparency and explainability of the model's decision-making process, which is crucial in real-world applications where trust and understanding of the model's outputs are of utmost importance.

1.2.4 Open-book QA

Typically, QA systems append the context documents to the input question to extract the answers. The context documents can be provided directly as in SQuAD dataset (Rajpurkar et al., 2016), in which case they are fed alongside the question to reader modules like BERT (Devlin et al., 2019), ALBERT (Lan et al., 2020) or other reader modules.

Another setting is that the system must also find the required documents from a more massive collection of documents or a small pool of candidate documents (e.g., ten). In the case of a huge collection, like the whole Wikipedia, systems would require information retrieval modules to retrieve the related documents. The top related documents are concatenated together (Beltagy et al., 2020) or they are used as context individually/in pairs (Tu et al., 2020b) alongside the question. The answering model then identifies the pieces of the context that best hold the answer for the question and generates a well-formed answer along with its probability. In the case of a one-by-one context, the answer with the highest score is selected as the final answer. Single-hop datasets for this setting include Natural Questions (Kwiatkowski et al., 2019) and TriviaQA (Joshi et al., 2017), while multi-hop datasets include HotpotQA (Yang et al., 2018) and 2WikiMultiHopQA (Ho et al., 2020).

The accuracy of those QA systems is bottlenecked by the quality of the information retrieval module's accuracy and the continuous availability of textual knowledge. This setting is called open-book QA.

In this thesis, we are concerned with the reasoning part of QA systems without considering information retrieval modules. Therefore, we focus on the open-book setting where the candidate documents are provided.

1.2.5 Closed-book QA

Recent studies have found that models could learn the factual knowledge present in the training data during their pre-training. Petroni et al. (2019) introduced a probe that evaluates how highly models rank the missing word from fact triples formed into cloze-style templates. They concluded that factual knowledge could be recovered remarkably well. Roberts et al. (2020) assessed the amount of knowledge stored in LMs in a closed-book setting. They concluded that LMs could attain competitive performance on open-domain QA benchmarks without reliance on context or external knowledge. In other words, those models could answer questions without requiring the context documents. This phenomenon would increase the speed of answering questions, as there is no information retrieval delay overhead. It would also eliminate the error propagation from wrongly retrieved documents and enhance mobility as only the model is required to answer questions. This setting is called closed-book QA.

Some recent well-known models demonstrating this phenomenon are ChatGPT and GPT-3 (Brown et al., 2020b). Their capabilities and potential make the setting of closed-book QA even more intriguing. Intuitively, as multi-hop questions are more complex, their performance lags behind simpler single-hop questions. A desirable trait of such systems is to have the performance of answering multi-hop questions close to the level of answering single-hop questions.

1.3 Motivation and Approach

Considering the importance of the QA task (Section 1.2.1) and that QA is an indicator of models' capacity for logical thinking, in this thesis, we focus on the

problem of QA. Improvements in QA translate into improvements in logical reasoning abilities, which would benefit numerous NLP applications.

As discussed earlier, depending on the utilization of context information, QA can be classified into two main categories: open-book QA and closed-book QA. While the closed-book QA setting appears advantageous, it does come with certain limitations. An example is that models learn factual knowledge from the data it was trained on. Any question about newer events after the training has concluded would be unanswerable without access to a real-time knowledge source. Another drawback is that such closed-book QA models are susceptible to what is called "hallucinations" where a model would output a made-up answer that looks legit.

In these cases, allowing access to a retrieved context about the question would permit the model access to more updated information. It would allow it to make concrete predictions about the answer instead of hallucinating answers, which is a key strength of open-book QA. Furthermore, open-book QA relieves the model from the burden of memorization and focuses on advancing the reasoning abilities of the model.

As both open-book and closed-book settings have their benefits and use cases, we must consider advancing and enhancing the precision of QA in both open-book and closed-book scenarios. In the case of open-book, given that single-hop models already exceed human-level performance, we focus on multi-hop questions. Previous studies attempted to utilize signals and supporting sentences from the context documents to find the correct answer. However, these systems did not exploit such information efficiently. As for the closed-book setting, previous work has studied this basic phenomenon without examining the internal mechanisms and without studying and considering improvements for multi-hop questions in this setting.

In this thesis, we aim to address the aforementioned shortcomings. We first start with multi-hop QA in an open-book setting. We seek to understand how the reasoning chain works and how the predicted supporting sentences affect answering the multi-hop question. We also improve answering multi-hop questions in the open-book setting by detecting useful hints in the contexts that benefit the multi-hop QA models (Alkhaldi et al., 2021).

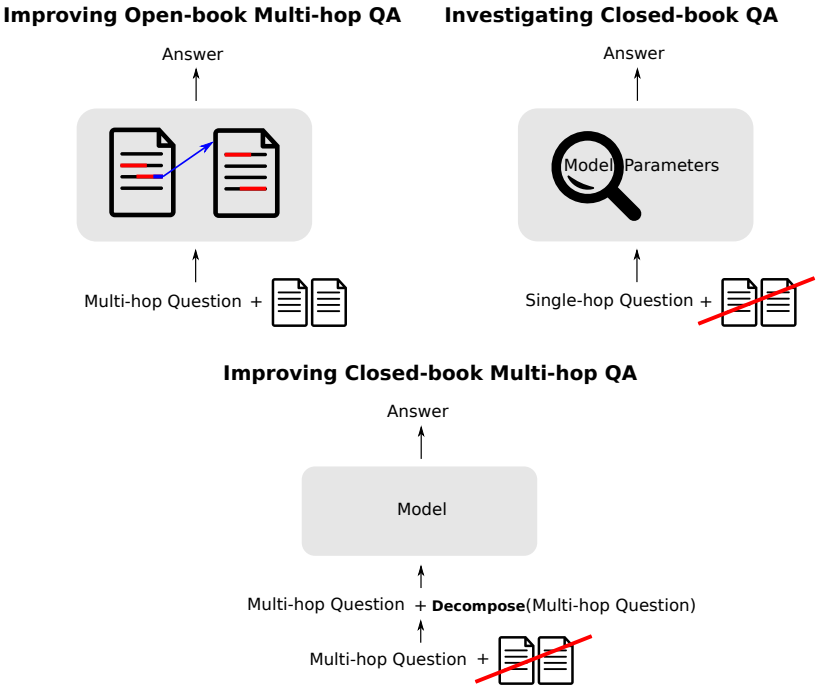

Figure 1.3: Overview of our work.

Subsequently, we shift our focus to closed-book QA. There is still a lot of mystery around this setting, and although studies have investigated the amount of knowledge learned (Petroni et al., 2019; Roberts et al., 2020), the inner workings and behavior of the internal parts of the models in this setting are poorly understood. Specifically, how do models memorize factual knowledge, and how is this knowledge stored inside the parameters of the model? What part of the Transformer block is the most crucial for answering questions? Do encoders and decoders have different roles? If the knowledge is concentrated in certain places, can we safely prune the other places of the model to reduce the model's size while keeping the performance? We conducted experiments to answer these questions (Alkhaldi et al., 2022).

As we learned more about closed-book QA, we move to enhancing it. Improving multi-hop QA in a closed-book setting is a novel endeavor; the studies on closed-book QA principally focused on simple, one-step reasoning questions, while studies on multi-hop QA were conducted in an open-book setting. We empirically

reveal the gap between single-hop and multi-hop QA in the closed-book setting. We proposed two methods for improvement using question decomposition. We also find useful insights on how to improve multi-hop QA in the closed-book setting further (Alkhaldi et al., 2023). We summarize our work in Figure 1.3.

1.4 Outline of the Thesis

In this thesis, we address three topics related to question answering: Open-book multi-hop QA, factual knowledge memory in closed-book QA, and answering multi-hop questions in a closed-book setting.

In Chapter 2, we present a detailed review of pivotal neural advancements and concepts that hold significant relevance not only in NLP as a whole but also specifically within the context of our thesis.

In Chapter 3, we present our work on improving multi-hop QA in an open-book setting. Models solving single-hop questions learned to be adept at pattern matching, therefore, inadequate to assess models' reasoning abilities. In contrast, multi-hop questions proved to be a better estimation of such abilities. We first introduce an overview and previous work on single-hop and multi-hop QA, then discuss the model architecture and our proposals. After finding supporting sentences related to the question, we flexibly focus on them to help guide the model's reasoning. We also show the benefit of tagging the entities bridging different paragraphs required for the answer.

In Chapter 4, we introduce our investigations and experiments on a Transformers model in a closed-book setting. The goal is to understand how knowledge is stored inside such a model and to analyze which parts are most responsible. We explain the task and the inner components of Transformers, and then we study visualized maps of gradients. We apply noise to different components to examine their contribution to finding the answer. Then we compare different strategies and propose a new one for assigning scores to attention heads inside the Transformer block. We present experiments using these scores to prune parts of the model that retain the most knowledge while keeping a comparable performance.

In Chapter 5, we focus on improving multi-hop QA in a closed-book setting

through question decomposition. With recent trends in language models acting as offline search engines, the ability to improve answering complex multi-hop questions became more beneficial. We describe the decomposition methods we use, and we show the results of experimenting on three datasets. We explain promising future research directions to further improve the accuracy.

In Chapter 6, we conclude this thesis with a comprehensive review of our work and explain future research prospects.

Chapter 2

Review of NLP Advancements

We provide an overview of the historical progression of neural network research up to the latest developments in deep learning within the context of NLP.

2.1 Language Modeling

A very basic task that is the core of today's NLP progress is the task of language modeling. A language model learns the probability of the next work given the previous words. Concretely, if $y_1, y_2, ..., y_n$ are tokens in a sentence, the probability of seeing those tokens in order is:

$$P(y_1, y_2, ..., y_n) = \prod_{i=1}^{n} P(y_i | y_1 ..., y_{i-1}) \tag{2.1}$$

They are trained on huge amounts of unlabeled text, which is significantly more accessible compared to labeled data. This extensive training allows the models to learn the language's nuances and semantics, and to understand the context of words and sentences. As an output of the training, these language models became powerful at recognizing if sentences are plausible or not and have many real-life applications, such as natural language generation and autocompletion. An example of autocompletion is in Figure 2.1.

Language models are also able to generalize to data that has not appeared in the training text. This is due to the fact that, unlike older non-neural models, words are converted into continuous space representations or vectors that are

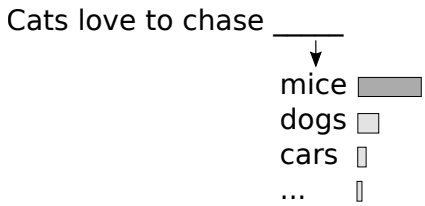

Figure 2.1: An example of how language models learn to distinguish plausable sentences.

called word embeddings. When those embeddings are learned and optimized, the embeddings of words would have the feature that similar words have closer embeddings. Models then learn that similar words can fit in a certain context, even if it had never seen those words in this specific context before.

2.2 Recurrent Neural Networks (RNNs)

Early feed-forward networks were insufficient to deal with language data as language is sequential in nature, while the feed-forward networks had no notion of sequentiality. RNNs (Mikolov et al., 2010) were introduced to mitigate this problem by incorporating loops in their architecture by feeding the output of certain layers into the same layers in subsequent time steps, hence referred to as recurrent. At every time step, it updates a memory (also referred to as its state). RNNs leverage their loops to produce continuous space representations for sequences of varying lengths, and because of their recurrent nature, they can be considered as deep with respect to the time sequence.

This architecture allows RNNs to capture long-distance relationships within a sequence and proved to be promising in predicting the subsequent word of a sequence in the language modeling task. An example of RNN is illustrated in Figure 2.2.

As the vanilla RNN architecture suffered from problems such as vanishing or exploding gradients, a new successor was introduced called long short-term memory network (LSTM) (Graves, 2012), which is a modification of RNNs. LSTMs facilitate improved retention of previous data in memory, which effectively ad-

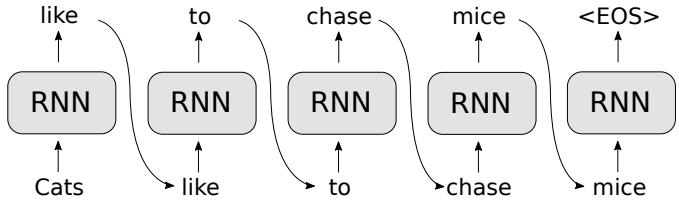

Figure 2.2: An example of how language models learn to distinguish plausible sentences.

dresses the vanishing or exploding gradient problem. It introduced multiple gates to the input, memory, and output layers. These gates serve to filter out irrelevant input or memory information while preserving essential data.

2.3 RNN-Based Encoder-Decoder Models

Considering the task of predicting a sequence given an input sequence, also referred to as sequence-to-sequence (Seq2Seq), a model of two RNNs corresponding to an encoder and decoder became popular. The encoder's job is to calculate the embeddings of the input or "source" sequence, while the decoder generates the output or "target" sequence using the encoded state.

An illustration of the flow is shown in Figure 2.3, and it goes as follows: The input sentence ending with an end-of-sentence <EOS> token is fed into the encoder. The last state of the encoder's RNN represents the contextualized embedding of the input sentence. This state is then fed into the decoder, and starting with a beginning-of-sentence <BOS> token, it starts to generate the output sentence until it predicts an <EOS> token.

Another variation of the above model is to feed the encoder state to the decoder at every time step, not only at the beginning, which allows it always to have the ability to look at the encoded input sentence information. This Seq2Seq model is effective for machine translation tasks and dialogue applications.

The above encoder-decoder model has a major limitation; it encodes the whole input sentence into a single fixed-length state which is used to decode the output. For longer sentences, it is difficult to squeeze all the information about the input sentence into a fixed-length vector. Therefore, the decoder struggles when it

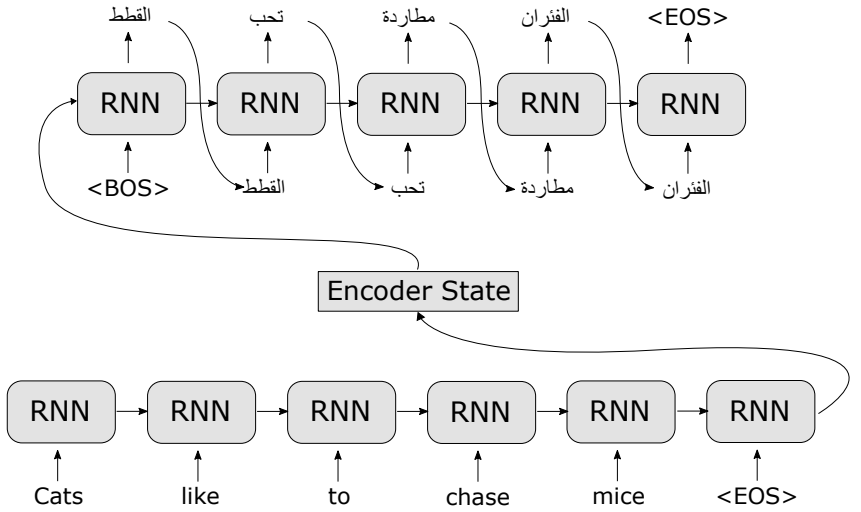

Figure 2.3: An example of an encoder-decoder model translating an English sentence into Arabic.

attempts to recover all input information from a single state vector.

To remedy this limitation, the attention mechanism was introduced (Bahdanau et al., 2015). When attention is applied to the encoder-decoder model above, the decoding process looks for the positions in the source sentence that are most relevant for the current time step. Then to predict the next word, in addition to the previously generated target words, it uses a combination of the context vectors associated with the related positions it found. This way, a context vector tailored for each time step is used, allowing the model to focus on different parts of the input adaptively each time.

2.4 Transformer

One of the drawbacks of RNNs is their sequential nature. For example, when an RNN wants to encode a sentence, it has to do so word by word, as it has to wait for the state of the previous timestep to calculate the next one. This is a bottleneck when training or inferencing.

The recently introduced Transformer (Vaswani et al., 2017) solves this problem by self-attention. It is a technique where the model attends to different parts

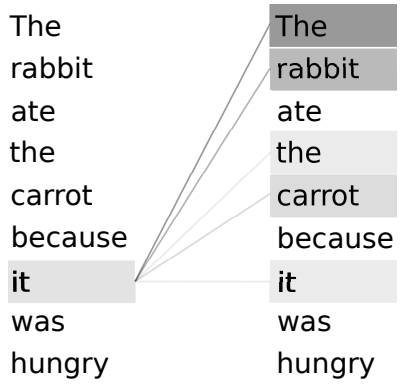

Figure 2.4: An example of a word attends to other words in self-attention.

within its input sentence to capture dependencies and relationships between words better. Figure 2.4 shows an example of applying self-attention to a sentence.

For self-attention to work, the input sequence is transformed into three different vectors: Query (Q), Key (K), and Value (V). These vectors are obtained by applying learnable linear transformations to the original input sequence. In other words, by multiplying the input embeddings with a learnable weight matrix. This step projects the input sequence into a higher-dimensional space, which enables the model to capture more complex relationships.

Then, to compute the similarity (or compatibility) between each query and each key element, the dot product is calculated between the Q of an element and the K of all elements in the sequence to represent attention scores. The products are scaled to avoid excessively large values to prevent instability during training. The resulting scores indicate how relevant each element is to the current query.

The computed attention scores are passed through a softmax function for normalizing the scores to ensure they sum to 1, representing a valid probability distribution. Those final attention weights are multiplied with their corresponding V vectors and summed. This produces the final attended (or contextualized) embeddings of the input sequence, which contain the most relevant information in the context of the current query.

This attention sub-layer can consist of multiple heads, where they have separate learnable parameters, and learn different features. They allow the model

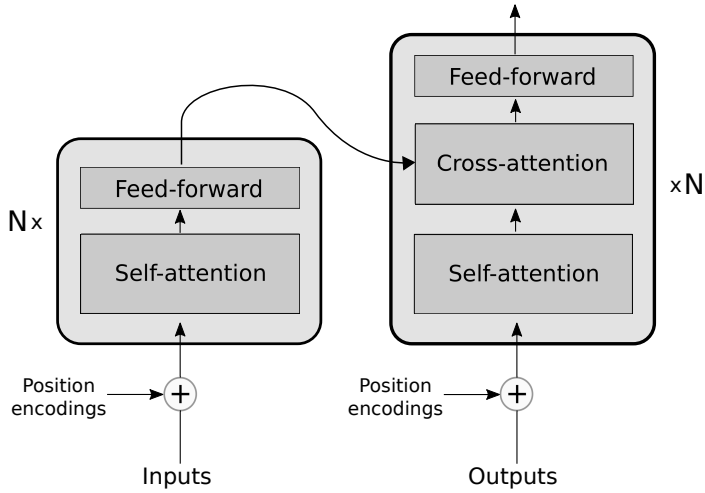

Figure 2.5: The Transformer's architecture.

to independently attend to different parts of the sequence in parallel, resulting in improved performance time and quality. The final multiple contextualized embeddings from different heads are concatenated and then passed through a linear transformation to get a single contextualized embedding.

The Transformer block consists of self-attention and feed-forward sub-layers. The blocks are stacked to form an encoder and decoder with N layers. The decoder additionally has a cross-attention sub-layer that attends to the encoded inputs. More specifically, this cross-attention calculates the Q vectors from the output sequence but takes K and V from the encoded input sequence in the encoder. An illustration of the Transformer's architecture is in Figure 2.5.

So far, Transformers cannot handle sequential information as they have no inherent sense of words' position or order. The input sequence is treated as a bag of words. Therefore, positional encoding is necessary, especially in tasks or languages where word order is important.

The Transformer architecture adds positional information by summing positional encodings for each input embeddings at the beginning of the encoder and decoder stacks. Every position index is mapped to different positional features. They use sine and cosine functions of different frequencies to assign the values of the position encodings because the model can easily distinguish the relative

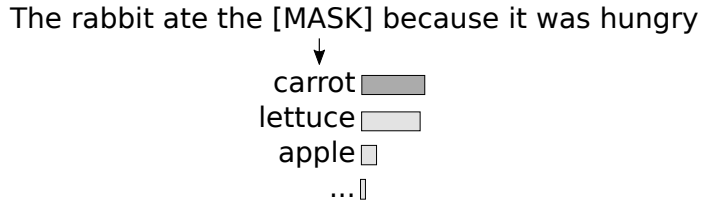

Figure 2.6: An example of the masked language modeling objective.

location based on the value of these functions.

2.5 Transformer-Based Encoder Models

Highly influential work built upon Transformer architecture is "Bidirectional Encoder Representations from Transformers" (BERT) (Devlin et al., 2019). It achieved many state-of-the-art results in many NLP tasks, such as natural language inference and question answering.

The main idea is that BERT pre-trains powerful bidirectional representations from raw, unlabeled text. It considers both the left and right context simultaneously across all layers. After pre-training on a language modeling objective, the model learns the language structures and can be fine-tuned to specialize in downstream tasks without significant changes in the architecture.

In typical language modeling tasks, the goal is to predict the next word given the previous words. However, bidirectional models are made to understand the surrounding text from both sides for richer contextual information, therefore, requiring a different modeling objective. They introduce a task called masked language modeling (MLM). In this task, certain words or tokens in the input sequence are randomly masked, and the model's objective is to predict the original masked words based on the surrounding context. An example is shown in Figure 2.6.

When pre-training BERT, another task known as next sentence prediction (NSP) was simultaneously employed. The main objective of NSP is to train a language model to predict whether one sentence naturally follows another in a given text corpus. In other words, the model learns to understand the relation-

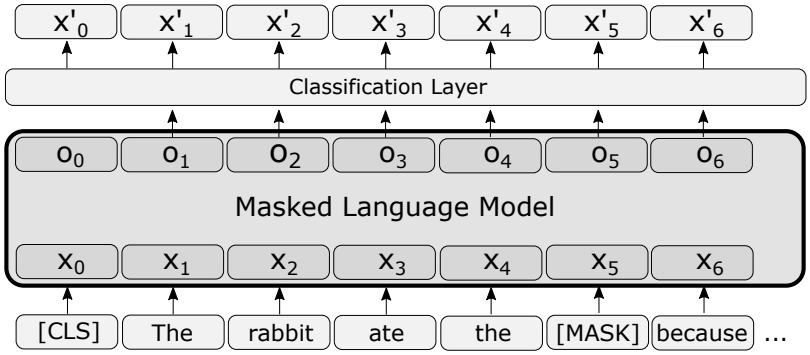

Figure 2.7: An overview of the BERT masked language model.

ship between two consecutive sentences in a document. Positive examples for training are pairs of consecutive sentences from the original corpus, while for negative examples, a second sentence is selected randomly from the corpus that does not follow the first sentence. This task helps comprehend the contextual flow of language and understand the connections between sentences.

The pre-train then fine-tune paradigm, also known as transfer learning, has emerged as a powerful approach for NLP and other machine learning. It is called transfer learning in the sense that it transfers the knowledge it learned from all the unstructured text it pre-trained on into the downstream tasks. This is especially useful as the downstream tasks often require labeled or annotated data that is significantly more expensive and harder to obtain. The transfer learning relieves the model from having to learn the language structure from the limited available labeled data and to only focus on the downstream task-related patterns. Transfer learning also helps with robust generalization across different domains.

The output of BERT can be utilized in various ways. Even without a classification layer on top, the output embeddings o_n capture valuable contextualized representations of the input sentence. For example, the special token `[CLS]` is usually added at the beginning of the sentence to capture the embedding of the whole sentence and to help with sentence-wise classification tasks. An illustration of the BERT paradigm is shown in Figure 2.7.

Those sentence embeddings can be employed for several tasks such as semantic similarity, where the model measures how similar are two sentences or documents

by computing the similarity between their embeddings. Another example is clustering, which involves using sentence embeddings in algorithms to group similar sentences or documents together.

By adding a classification layer on top, BERT can be fine-tuned for handling other document-level tasks such as sentiment analysis (determining if the sentence is positive, negative, or neutral), spam detection, or topic classification. BERT can also be used for word or token-level classification tasks. For example, named entity recognition (NER), where the model learns to detect named entities like person names, organizations, and locations. One way to do that is to predict for each token the entity class and if it's the beginning token in a named entity (B), intermediate (I), or not a named entity (O).

Another important task that BERT can solve is question answering. By utilizing its contextual understanding, BERT is capable of predicting the start and end positions of the answer span within a provided context paragraph. Based on the `[CLS]` token, it can also predict if the paragraph contains an answer or not.

It is worth noting that variations of BERT improved on it in various ways. For example, RoBERTa (Liu et al., 2019) enhanced the performance by training with more optimized hyperparameters such as batch size and training steps. ALBERT (Lan et al., 2020) uses parameter-reduction techniques to reduce BERT's size, allowing it to scale better. This is effective as BERT's model parameter size has been shown to impact the performance directly.

BERT-based models explained so far have one strict limitation, the maximum context length. Models commonly can only process up to 512 sequence lengths, which makes them inadequate for handling long documents. Naively increasing the context size with the same architecture would result in quadratically growing memory and computational requirements because of the self-attention components.

Various techniques are introduced to allow increasing the context length without the computational overhead. Some work (Beltagy et al., 2020) incorporates special tokens that represent sentences and only does self-attention with a limited context window called local attention. Then, in addition to the local attention, all special tokens attend to each other in what is called global attention. This allows

the model to reach information from far-away contexts that are summarized in the embeddings of those global tokens.

Another method (Wang et al., 2020) achieves similar improvement by compressing the attention matrix using low-rank factorization, which reduces the number of parameters and computations required during attention calculations. By significantly reducing memory and computation overhead, they enable transformer models to process longer sequences more efficiently without significantly sacrificing performance.

The increased sequence length limit is useful for several cases like document classification or document summarization. The benefit does not only come from the ability to let words focus on a broader context but also as a more efficient approach than splitting long documents, simplifying preprocessing, and allowing parallelism.

2.6 Transformer-Based Encoder-Decoder Models

Fine-tuning BERT-based models requires a customized classification layer, tailored training loss objective, and hyperparameters to handle downstream tasks. Avoiding these constraints, an encoder-decoder model called Text-to-text Transfer Transformer (T5) was introduced (Raffel et al., 2020).

The T5 was pre-trained in an objective similar to masked language modeling, where they randomly masked 15% of the input and trained the model to generate the masked words. After pre-training, they transformed all NLP tasks into textual input and output and continued the training. This greatly unifies the training across tasks, as models now do not require specialized architectural changes or hyperparameter changes to adapt to different tasks. Another benefit is the ability to easily switch tasks by prompting. For example, starting the text with “Translate this from X to Y language: ”, the model performs machine translation. When the prompt starts with “Summarize this document: ...” document summarization is done.

Another encoder-decoder model is BART (Lewis et al., 2020). They employ a different pre-training techinque in which they corrupt the input and teach the

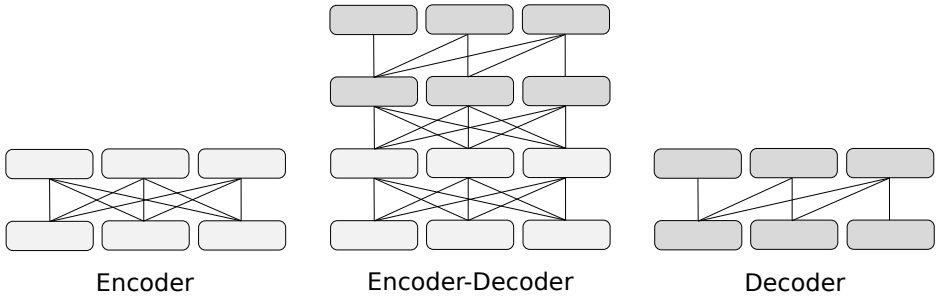

Figure 2.8: An overview of attention connections in different model architectures.

model to predict the original input. Corruptions include randomly shuffling the sentences' order, token masking and deletion, and span masking. However, unlike T5, the fine-tuning stage still used classifiers on top of the final embeddings for different tasks like sequence and token classification.

2.7 Transformer-Based Decoder Models

Another family of models is decoder-only models. These generally excel at generating natural text and are often called generative pre-trained Transformers (GPT). One difference between encoders and decoders is the tokens on which they apply attention. In encoders, it is typically done bidirectionally, that is, from left-to-right and right-to-left. For decoders, they only attend on the tokens already generated from the left, in other words, left-to-right only. Figure 2.8 shows an illustration of attention connections in different model architectures.

Because decoders are unidirectional and attend only to left tokens, they do not undergo pre-training with masked language modeling. Instead, they are pre-trained with a standard language modeling objective, as elaborated in Section 2.1.

As the effect of a model's size was shown to correspond to its performance, researchers have been scaling models from GPT-2 (Radford et al., 2018), which has 1.5 billion parameters to GPT-3 (Brown et al., 2020b) of a staggering 175 billion parameters. Although managing and supporting models of this magnitude is challenging and costly, they have demonstrated remarkable capabilities.

Few-shot prompt
Input: Analyze the sentiment of movie reviews, only outputting the sentiment word. Review: This movie is fantastic! Sentiment: Positive Review: I disliked the film's ending. Sentiment: Negative Review: The acting was brilliant, but the plot was weak. Sentiment: Neutral Review: The movie was confusing. Sentiment:
Output: Negative

Table 2.1: Example of few-shot prompting.

One emerging powerful capability is the ability to learn from context, also referred to as *in-context learning*. This paradigm is helpful if there is a new task we want the model to solve, but we have not trained it on that task before. To teach the model this task without additional training or parameter updates, we can explain the task and give examples of the desired input and output, and the model can solve it. This method is called few-shot prompting because it only depends on a few examples. It is also useful even if the model knows the task but is interested in guiding it to respond in a specific way or format. An example is shown in Table 2.1.

Another ability is zero-shot prompting, where the model is presented with an unfamiliar prompt that is not included in its training data, but the model can still generate the desired output.

In regards to explainability, one advantage of a large enough model like GPT-3 is that it can explain its reasoning process, or chain-of-thoughts (CoT), in a human-readable way. It's an improvement over the long-held view of black-box neural models. When instructed with "Let's think step-by-step." after the desired

question, the generated explanation can even help the model itself with reasoning, especially in math-related questions. However, it is still prone to errors and hallucinations in explanations and the final answer.

Chapter 3

Enhancing Multi-hop QA in Open-Book Setting

With the help of the detailed annotated question answering dataset HotpotQA, recent question answering models are trained to justify their predicted answers with supporting facts from context documents. Some related works train the same model to find supporting facts and answers jointly without having specialized models for each task. The others train separate models for each task, but do not use supporting facts effectively to find the answer; they either use only the predicted sentences and ignore the remaining context, or do not use them at all. Furthermore, while complex graph-based models consider the bridge/connection between documents in the multi-hop setting, simple BERT-based models usually drop it. We propose FlexibleFocusedReader (FFReader), a model that 1) Flexibly focuses on predicted supporting facts (SFs) without ignoring the important remaining context, 2) Focuses on the bridge between documents, despite not using graph architectures, and 3) Jointly learns predicting SFs and answering with two specialized models. Our model achieves consistent improvement over the baseline. In particular, we find that flexibly focusing on SFs is important, rather than ignoring remaining context or not using SFs at all for finding the answer. We also find that tagging the entity that links the documents at hand is very beneficial. Finally, we show that joint training is crucial for FFReader.

The task of question answering (QA) is to find an answer to a natural language

question from a given text (Rajpurkar et al., 2016). With the goal of training systems to apply reasoning and inference on text, and measuring their performance quantitatively, many datasets have been introduced. One of the early large-scale ones includes SQuAD (Rajpurkar et al., 2016), where questions were designed to be answered from a single paragraph, and thus called single-hop QA. Systems achieved human performance, without achieving the sought-after reasoning skill (Clark et al., 2018; Yang et al., 2018), as questions could mostly be answered from a single sentence which encouraged the systems to focus more on matching information between the question and text (Welbl et al., 2018).

To stimulate models to use more complex reasoning, the task of multi-hop QA was introduced. In this task, reasoning over multiple documents is required to find an answer (Welbl et al., 2018). A popular dataset for this task is HotpotQA (Yang et al., 2018) which, in addition to the answer, has per-sentence annotations for which sentences are supporting facts (SFs). The goal of asking models to predict answers and supporting facts is to encourage models to explain how they reach an answer.

Models that are recently introduced for this task generally divide into two architectures: Graph-based models that use some form of Graph Neural Networks (GNNs) (Scarselli et al., 2009) like (Fang et al., 2020; Tu et al., 2020a; Shao et al., 2020), and non graph-based models that are a pipeline of BERT-based (Devlin et al., 2019) models like (Glass et al., 2020; Beltagy et al., 2020). Some models use predicted answers to find the SFs (Asai et al., 2020; Shao et al., 2020), or use SFs prediction as a second task on the same model that predicts the answers (Fang et al., 2020; Qiu et al., 2019) (not using any predicted SFs as input to find the answer; i.e. Unfocused Reader), or only feed the predicted SFs to the answering model, ignoring the remaining context (Glass et al., 2020; Groeneveld et al., 2020) (i.e. Strict Focused Reader).

We think that the healthy way to find an answer is by reasoning on the SFs, however ignoring the remaining context might leave out important information, either due to annotation problems or sub-optimal SFs prediction. It will also limit the model to answer only from within the SFs. We propose a novel way of focusing on the SFs; we tag them while keeping the remaining context, thus allowing our

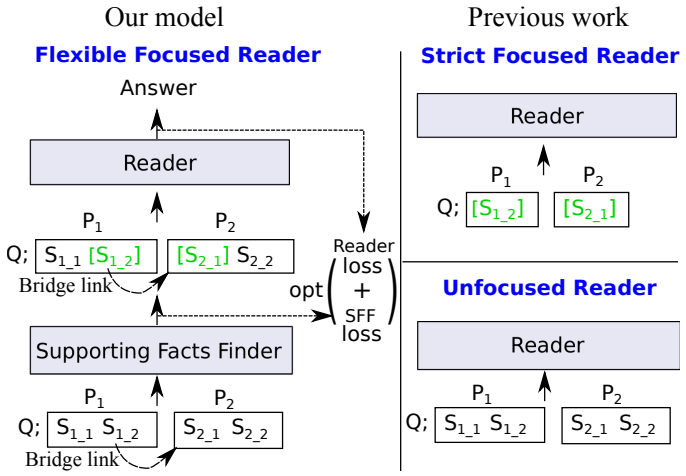

Figure 3.1: High-level comparison between our proposed model with similar non-graph-based related work models. With a question and a paragraph pair as input, our model uses SFs annotations effectively and does not ignore the bridge link between the paragraphs. The reader module is jointly trained with the SFF module. Here, $\texttt{S}_{\texttt{j\_i}}$ represents sentence i of paragraph j. Strict Focused Reader means that the model only uses the predicted SFs as input to the reader, while Unfocused Reader does not use SFs to predict the answer.

reader model to predict correct answers even from outside of the SFs as we show in Section 3.6.

We also identify and tag entities that link paragraph pairs, as we find it is an important cue that non-graph-based models lack. Models incorporate complex[1] GNNs to include this link, while we keep our model simple and still include it.

In order to benefit from both sub-task signals, previous models (Fang et al., 2020; Tu et al., 2020a; Beltagy et al., 2020) have jointly trained the same model to both answer and predict SFs. As observed by Beltagy et al. (2020), this training method hurts the reader's performance, likely due to less capacity and not being specialized in each sub-task. To avoid this, we train a specialized reader and an

[1]Complexity can be in graph construction and keeping information of nodes and edges for every example (which includes recognizing named entities in some models like (Fang et al., 2020)), or in performance, where the run overhead depends on the number of nodes/edges, and the number of message passing iterations

SF finder modules jointly.

We show a comparison of our work and previous work in Figure 3.1. Our final model achieves clear improvement over the baseline and non-graph-based models and achieves comparable results with the more complex state-of-the-art models. We show that when jointly trained, flexibly focusing on SFs benefits from the SFs annotations for finding the answer without limiting the answer space. Our proposal of tagging SFs to focus on them (instead of only using them or ignoring them at all) is a general recommendation for QA tasks, and it can be applied even to datasets that do not have SFs supervision like SQUAD (Rajpurkar et al., 2016); we can still predict SFs and use them.[2] Furthermore, we show that identifying and tagging bridge entities is important for multi-hop QA, and we expect this finding to be applicable to any other Wikipedia-based multi-hop QA dataset where we can benefit from the hyperlinks between articles like WikiHop (Welbl et al., 2018) and HybridQA (Chen et al., 2020).

3.1 Related Work

3.1.1 Single-hop QA

Questions in single-hop QA datasets like SQuAD (Rajpurkar et al., 2016), WebQuestions (Berant et al., 2013a), SimpleQuestions (Bordes et al., 2015) and NaturalQuestions (Kwiatkowski et al., 2019) can be answered using a single paragraph or document as context. Since the introduction of Transformers (Wolf et al., 2020) and BERT (Devlin et al., 2019), best-performing systems have been extensions of such pre-trained models like RoBERTa (Liu et al., 2019), ALBERT (Lan et al., 2020) and ELECTRA (Clark et al., 2020), with a typical span prediction head on top. The task of mere span prediction from a single document was not enough to let the models learn to answer more complex questions that require reasoning across multiple documents, and so multi-hop QA was considered as the next step.

[2]The investigation of such application is left for future work.

3.1.2 Multi-hop QA on Knowledge Bases

Some datasets focus on enabling QA over knowledge bases (KBs). The WebQuestions semantic parses (WebQSP) dataset (Yih et al., 2016) provides semantic parses of questions answerable from Freebase (Bollacker et al., 2008) and they require up to 2-hop reasoning. ComplexWebQuestions (CWQ) (Talmor and Berant, 2018) provides crowd-sourced compositional natural language questions answerable from Freebase, and also requiring multi-hop reasoning. Some models for multi-hop knowledge base QA (KBQA) decompose complex questions into simpler questions and do reasoning depending on intermediate answers (Talmor and Berant, 2018) or use two networks for finding answers from the KB and deciding better intermediate reasoning (He et al., 2021). Knowledge bases, however, can be hard to maintain and noisy to generate automatically.

3.1.3 Multi-hop QA on Text

Multi-hop QA over text datasets like HotpotQA (Yang et al., 2018) and WikiHop (Welbl et al., 2018) were introduced to encourage systems to learn more complex reasoning as the pieces of evidence to answer a question are scattered among different documents, as opposed to single-hop QA datasets and KB-based datasets. HotpotQA also includes SFs annotations to encourage models to explain their reasoning.

Graph vs Non-graph Based Models

Multi-hop QA requires the model to hop between documents that are usually connected by a link to find the answer. This bridge connection is ignored by non-graph-based models (Glass et al., 2020; Beltagy et al., 2020) where they encode the concatenated question and context and perform classifications on top of the transformers (Vaswani et al., 2017) output. To make use of the connection between sentences and documents, some models incorporate GNNs (Kipf and Welling, 2017; Velickovic et al., 2018). SAE (Tu et al., 2020a) predicts answers directly from transformers output but applies GNNs on top of the generated sentence embeddings to predict SFs. HGN (Fang et al., 2020) constructs a hierarchical graph

of multiple levels of granularity (paragraphs, sentences, and entities), and predicts answers and SFs on top of this GNN. Shao et al. (2020) argue that graph structure might not be necessary for multi-hop question answering when pre-trained transformers are fine-tuned, and that graph attention can be considered as a special case of self-attention. In our model, we follow the simplicity of Longformer (Beltagy et al., 2020) but without sacrificing the bridge connection signal between documents. BigBird (Zaheer et al., 2020) is a very similar model to Longformer, with the main difference being the addition of either random attention or external tokens for global attention.

Utilization of Supporting Facts Annotations

HGN and DFGN (Qiu et al., 2019) models use SFs annotations implicitly in a multi-task setting. QUARK (Groeneveld et al., 2020) and TAP2 (Glass et al., 2020) explicitly use predicted SFs as the only context available to the answer-finding model. This strictness is harmful as not only the accuracy of the SF prediction is not optimal, but also the golden annotation itself has the problem of leaving out important related sentences, as discussed in Section 3.6.10. We circumvent this by tagging the predicted SFs while keeping the remaining context.

Joint Training

Models that apply joint training like HGN, DFGN, QFE (Nishida et al., 2019) and Longformer 1-stage version, do it in a multi-task way on the same model. The official HotpotQA baseline model (Yang et al., 2018) adds several layers on top of its SFs prediction module but its model still shares the same low-level representations, therefore joint training still happens on the same model. As Longformer's results show, the 2-stage mode (2 separate models for answer and SFs prediction) is better than the 1-stage mode, because having separate models for each task means more capacity and specialized models. In our model, we jointly train separate specialized models. We also experiment with additional settings like different ratios of loss combination and pre-initialization for joint fine-tuning as discussed in Section 3.6.8 and 3.6.6.

Question Type	train	dev
Comparison	17,456	1,487
Bridge	72,991	5,918
Total	90,447	7,405

Table 3.1: Statistics showing the amount of question types for each data split in HotpotQA. Note that the test split is hidden in the distractor setting.

3.2 The HotpotQA Distractor Task

3.2.1 Task Description

We use the HotpotQA dataset (Yang et al., 2018), which has two settings: Distractor and fullwiki. In this paper, we focus on the distractor setting as it is only concerned with the reader model part of the problem, not the information retrieval part. In the distractor setting, 10 paragraphs from 10 different Wikipedia documents are given, and only 2 paragraphs are related to the question to be answered and explained. The two sub-tasks are: 1) Answer prediction, and 2) Supporting facts prediction. They are evaluated with exact match (EM) and partial match (F1) metrics. The final performance is evaluated with a joint EM and F1 score.

The questions in this dataset have two types: "Bridge" and "Comparison." "Bridge" questions are anchored around a bridge entity (i.e., a hyperlink) that connects the paragraphs, while "Comparison" questions are about two paragraphs, not necessarily connected by a link. Table 3.1 shows statistics about the percentage of each question type in the dataset, and we see that "Bridge" type questions are the majority.

3.2.2 Data Preparation

Each question with its 2 gold and 8 distractor paragraphs is considered an example. In training, we generate 3 paragraph pairs for each example: 2 gold, 1 gold 1 distractor, and 2 distractor paragraphs. In evaluation, we consider all possible paragraph pairs, and we choose the pair with the highest score as shown in Section 3.3.4.

As alternative preparation settings, we also experiment with only using 2 gold paragraphs, or with 2 gold and 1 gold and 1 distractor without 2 distractor paragraphs, but we find that the performance degrades in both cases. Related work methods are either not concretely explained or not applicable to our model; Longformer (Beltagy et al., 2020) inputs 10 paragraphs at once, while HGN (Fang et al., 2020) selects several paragraphs using string-matching heuristics together with a trained ranker when needed, and uses them for training. SAE (Tu et al., 2020a) uses only gold paragraphs and uses a trained ranker to retrieve the top 2 paragraphs.

3.3 Model Flow

We show an overview of our model in Figure 3.2. We first explain the model flow, then talk about our contributions in Section 3.4.

3.3.1 LongRoBERTa

The basic unit in our model is a long version of RoBERTa (Liu et al., 2019), which is constructed in a similar way to Longformer, with the difference being in the maximum token length. LongRoBERTa and Longformer are different from plain RoBERTa mainly because of using global attention, which we set only on selected tokens as explained in Longformer (Beltagy et al., 2020). Following Longformer authors' instructions of continuing pretraining after the construction of a longer RoBERTa, we pre-trained on Wikitext103 (Merity et al., 2017) for 3k steps. Longformer inputs 10 paragraphs at once as context with a maximum token length of 4,096. This can be noisy as paragraphs from different documents are not coherent, and it is difficult to focus on links between paragraphs as there can be many links. Therefore, we only consider 2 paragraphs at a time with a maximum token length of 1,024, reducing noise and allowing us to do bridge tagging. LongRoBERTa is used for the reader module and the SFs finder module. We do not use the plain 512 tokens RoBERTa to avoid truncating or complex splitting into windows. We show the effects of using a plain RoBERTa versus LongRoBERTa in Section 3.6.7.

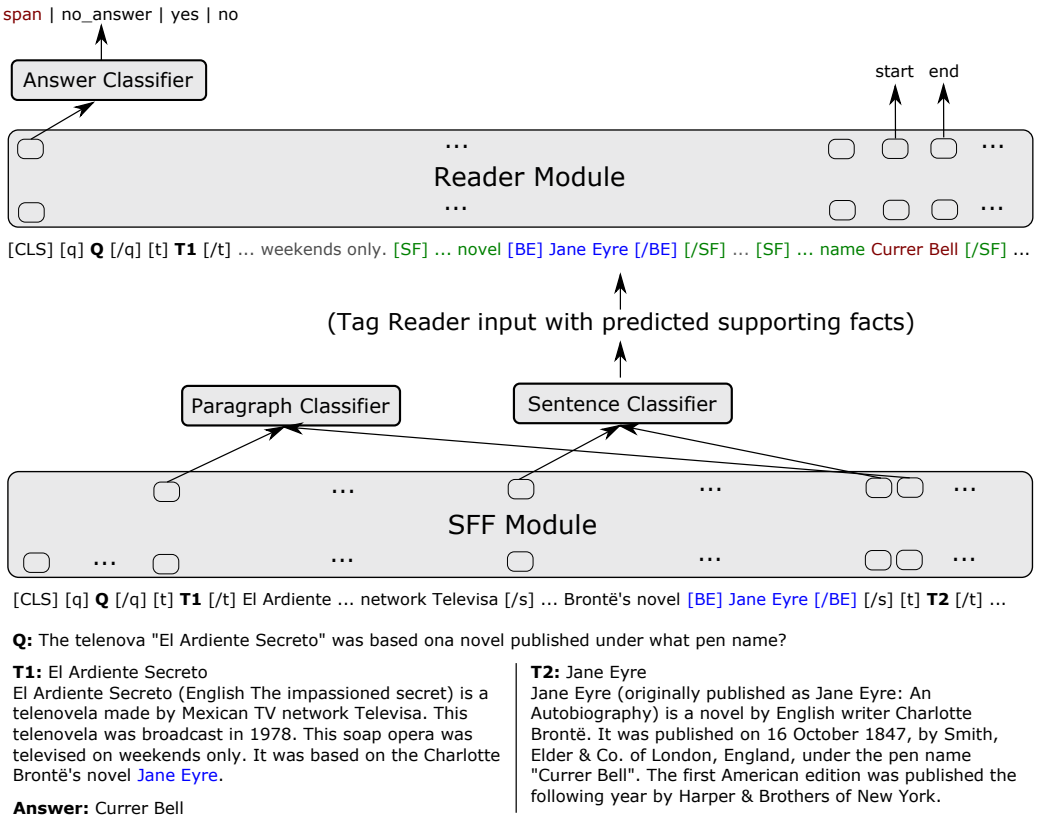

Figure 3.2: A detailed diagram that shows our model's architecture. The question, paragraphs, and their titles are concatenated, with the bridge entity tagged with a [BE] tag (tokens are shown in blue color). The [t] tags represent the paragraphs and are passed to the "Paragraph Classifier" to calculate the score P_j of paragraph j as in Eq. 3.4. The [/s] tags represent the sentences and are passed to the "Sentence Classifier" that classifies each sentence as an SF or not. In the input to the reader module, the sentences that are predicted to be SFs are tagged with [SF] tags (tokens are shown in green color). The start and end of a candidate answer span are then predicted, along with a classifier on the [CLS] token deciding whether to use this predicted span (tokens are shown in red color), give a yes/no answer, or decide that no answer is available. Note that Joint training is not depicted in this figure for clarity.

3.3.2 Supporting Facts Finder (SFF) Module

We concatenate the question with the paragraph pair after tagging them with special tokens as follows: "`[CLS] [q] question [/q] [t] title1 [/t] sent1_1 [/s] sent1_2 [/s]...[t] title2 [/t] sent2_1 [/s]...[SEP]`" where special tokens `[q]` and `[/q]` are question boundaries, `[t]` and `[/t]` are paragraph title boundaries, and `[/s]` is sentence ends. We consider `[t]` to represent the whole paragraph, and `[/s]` to represent the sentence, and we only assign `[t]` and `[/s]` tokens to have global attention, all following Longformer. Also similar to Longformer, we apply two-layer feedforward networks on top of the LongRoBERTa output of `[t]` and `[/s]` tokens to calculate the binary scores of the relatedness of paragraph j ($\mathbf{o}_{para\_j}$) and sentence i of paragraph j ($\mathbf{o}_{sent\_j\_i}$):

$$\mathbf{o}_{para\_j} = MLP_1(\mathbf{P}_j) \tag{3.1}$$

$$\mathbf{o}_{sent\_j\_i} = MLP_2(\mathbf{S}_{j\_i}) \tag{3.2}$$

where MLP($\cdot$) denotes Multi Layer Perceptron (MLP), $\mathbf{P}_j$ is the embedded output of token `[t]` of paragraph j, and $\mathbf{S}_{j\_i}$ is the embedded output of token `[/s]` of sentence i of paragraph j. From the two scores in $\mathbf{o}_{para\_j}$ (related and not related), we denote P_j to be the logit of paragraph j being related to the question. We use cross entropy to get the final loss of this module, SFF_{loss}:

$$SFF_{loss} = CE(\mathbf{o}_{para}, \mathbf{y}_{para}) + CE(\mathbf{o}_{sent}, \mathbf{y}_{sent})$$

where $\mathbf{o}_{para}$ and $\mathbf{o}_{sent}$ are vectors of binary scores for every paragraph and sentence, $\mathbf{y}_{para}$ and $\mathbf{y}_{sent}$ are the labels of the paragraph being related or sentence being a supporting fact, respectively. CE() represents the cross-entropy loss function. Note that in $\mathbf{y}_{para}$, a paragraph is given label `1` if it contains at least 1 SF.

3.3.3 Reader Module

We pass the prepared input without `[/s]` tokens to another instance of LongRoBERTa, with global attention for all question tokens only. Following a typical QA model (Devlin et al., 2019), we predict the start and end tokens from context

outputs and apply a multi-class classifier on top of the embedded output of the `[CLS]` token ($\mathbf{H}_{CLS}$) with 4 classes as:

$$\mathbf{o}^i_{answer} = \mathbf{W}_a \mathbf{H}_{CLS}, \tag{3.3}$$

where $\mathbf{W}_a$ is a learnable weight matrix, and for paragraph pair i, $\mathbf{o}^i_{answer} \in \mathbb{R}^{1\times 4}$ represents the logits of the 4 classes: $[span, no\_ans, yes, no]$, where they mean the answer is the span predicted by start and end logits, or no answer is found in the current paragraph pair, or yes and no answers to "Comparison" type questions, following Asai et al. (2020). The final loss, $Reader_{loss}$ is calculated as:

$$\begin{aligned} Reader_{loss} = & \, CE(\mathbf{o}_{start}, \mathbf{y}_{start}) \\ & + CE(\mathbf{o}_{end}, \mathbf{y}_{end}) \\ & + CE(\mathbf{o}_{answer}, \mathbf{y}_{answer}) \end{aligned}$$

where $\mathbf{o}_{start}$ and $\mathbf{o}_{end}$ are the logits of the start and end positions of the predicted span in the range of all possible indices. $\mathbf{y}_{start}$, $\mathbf{y}_{end}$ and $\mathbf{y}_{answer}$ are the labels of the start, end positions, and the answer class, respectively.

3.3.4 Evaluation Time Paragraph Pair Selection

In evaluation time, we select the paragraph pair i that has the highest score $Pair_i$ as follows:

$$Pair_i = \sum_{j \in S_i} P_j - no\_ans^i_{logit} \tag{3.4}$$

where S_i is the set of the selected paragraphs in pair i according to the binary score defined in Eq. 3.1, $no\_ans^i_{logit}$ is the no answer logit for paragraph pair i as explained in Section 3.3.3.

3.4 Our Proposal

3.4.1 Overview

With the flow described in Section 3.3, we add our contributions that manifest in 1) Flexibly focusing on SFs, that are usually not used in predicting the answer,

or used in a strict, non-flexible way, 2) Tagging bridge entities ("Jane Eyre" in the example) that are usually ignored with non-graph-based systems, and 3) Joint training the SFF module with the reader module which proves to be crucial for the flexibly focusing on SFs. We explain each contribution in detail in the following sections.

3.4.2 Flexibly Focusing on Predicted SFs

When preparing the input for the reader module (Section 3.3.3), we focus on predicted SFs by tagging related sentences as "`[SF] sent1_2 [/SF]`," while keeping the tagged bridge entities. We call our model that uses this technique: "Flexible-FocusedReader" (FFReader). We consider the baseline to be "UnfocusedReader" which is similar to our model except that it does not tag any SFs. We also compare against "StrictFocusedReader" (SFReader), where only predicted SFs are included as context, ignoring remaining sentences.

3.4.3 Bridge Entity Tagging (BET)

We use the raw Wikipedia text with hyperlinks to extract links and their indices and save everything in an indexed database. For each paragraph in the input pair, we retrieve available links and match them against the other paragraph in the pair. If the paragraphs are linked, we tag the tokens of the bridge entity as `[BE] hyperlink text [/BE]`. This tagging is important mostly in "Bridge" question types. Since they are the majority (as shown in Section 3.2.1), the importance of this proposal is well reflected in practice (Section 3.6).

3.4.4 Joint Training

We jointly train the SFF and reader modules by combining their losses as follows:

$$Loss = SFF_{loss} * \lambda + Reader_{loss} * (1 - \lambda) \tag{3.5}$$

where $\lambda \in [0, 1]$ is a hyperparameter to control the importance of each module. We find in our experiments that the optimal λ is 0.5. Detailed λ comparison can be found in Section 3.6.8. While this joint training is crucial for the reader

module, we find that it does not improve the SFF module. In fact, it degrades its performance; therefore, in our final model, we use a separately trained SFF module + the jointly trained reader module. We also experiment with a joint training setting where we initialize the modules with separately trained ones and only fine-tune them, but we did not gain major improvement. We give more details in Section 3.6.6.

3.5 Experiments

3.5.1 Implementation Details

We use Transformers library (Wolf et al., 2020) in our implementation with *base* and *large* versions of RoBERTa (Liu et al., 2019). Both *base* and *large* versions are extended to have max tokens of 1,024 (originally 512) following the instructions of Beltagy et al. (2020) for building "long" version of pre-trained models. The difference between them is: 12-layer, 768-hidden and 12-heads for *base* vs 24-layer, 1,024-hidden and 12-heads for *large*. For both *base* and *large*, we use the Adam optimizer with warmup equal to 0.1 of the total steps and linear decay. We use a learning rate of 5e-5 with a batch size of 32. For *base* joint experiments, we use a learning rate of 2e-4 with a 512 batch size with Adasum (Maleki et al., 2021). We train *base* versions for 10 epochs, while *large* for 20 epochs, and use $\lambda = 0.5$ for both versions of joint experiments.

3.5.2 Results

We show the test results of our *large* model on the distractor setting of the HotpotQA dataset in Table 3.2. "Ans" and "Sup" refer to the tasks of finding answers and supporting facts, respectively. "Joint" refers to the joint evaluation where for each question, the answer *and* the supporting facts should be correct to get the EM score, or partially correct to get the F1 score. We see that our model outperforms all non-graph-based models and several graph-based ones for the "Ans" task, and achieves comparable scores with sophisticated graph-based models on both tasks.

Model Categories	Ans		Sup		Joint	
	EM	F1	EM	F1	EM	F1
Non graph-based						
TAP2 (Glass et al., 2020)	64.99	78.59	55.47	85.57	39.77	69.12
Longformer (Beltagy et al., 2020)	68.00	81.25	63.09	88.34	45.91	73.16
ETC-large* (Zaheer et al., 2020)	68.12	81.18	**63.25**	**89.09**	**46.40**	73.62
FFReader-large (Ours)	**68.89**	**82.16**	62.10	88.42	45.61	**73.78**
Graph-based						
SAE-large (Tu et al., 2020a)	66.92	79.62	61.53	86.86	45.36	71.45
SEGraph †	68.03	81.17	61.70	87.43	44.86	72.40
C2F Reader (Shao et al., 2020)	67.98	81.24	60.81	87.63	44.67	72.73
BFR-Graph †	**70.06**	**82.20**	61.33	88.41	45.92	74.13
HGN-large (Fang et al., 2020)	69.22	82.19	**62.76**	**88.47**	**47.11**	**74.21**
Unknown arch.						
GSAN-large †	68.57	81.62	62.36	88.73	46.06	73.89
SpiderNet-large †	70.15	83.02	**63.82**	**88.85**	47.54	74.88
AMGN+ †	**70.53**	**83.37**	63.57	88.83	**47.77**	**75.24**

Table 3.2: Test scores on the distractor setting of HotpotQA. We split the top models in the leaderboard into categories based on their architectures. **Ours** denotes our FFReader-large model. Models with † sign lack any details other than the test scores on the official leaderboard (https://hotpotqa.github.io/) as of January 28th, 2021. (* ETC-large is the name reported on the official leaderboard, but the actual full name is BigBird-ETC.)

We think the reason Longformer and BigBird-ETC have better EM scores (and F1 for BigBird-ETC) in predicting SFs in the "Sup" task is probably because when they concatenate all 10 paragraphs, they have a lot of negative sentences to look at and train on. One possible future work is to experiment with even more negative sampling settings than discussed in Section 3.2.2.

In the "Sup" task in Table 3.2, EM and F1 scores represent the precision and recall of SFs, respectively. Even though in the training data, we make sure the answer is in the gold SFs, our FFReader is flexible and does not require the answer to be in the predicted SFs, as we show in Table 3.11- Question 2. Therefore, we do not suffer from the necessity of having a huge recall on the SFs to make sure the answer is included, as in an SFReader. For example, the SFReader-based model TAP2 (Glass et al., 2020) selects SFs using a fixed threshold to make sure their recall (F1 score) is high since it is more important than precision (EM score) because if the SF containing the answer is missed, there is no way to answer correctly.

In our SFF module, we do not actually explicitly control the threshold for selecting an SF, we follow Longformer by predicting a binary score for each sentence [score_0, score_1]; if score_1 is larger than score_0, the sentence is considered an SF. Therefore, there is no precision/recall balance hyper-parameter that we tune.

Related work of graph-based architectures usually applies Graph Neural Networks (GNNs) (Velickovic et al., 2018) on top of a Transformers (Vaswani et al., 2017) model to do SFs prediction and answering. To use our flexible focusing on SFs proposal, they need to first identify the SFs, then answer. Therefore they need two separate versions of their model for each sub-task. This would mean their final architecture would be as follows: "(Transformer model + GNNs) → SFs + (Transformer model + GNNs) → Answer." Our model however would just be "(Transformer model) → SFs + (Transformer model) → Answer." In this sense, the double addition of GNNs is an added complexity and performance cost.[3]

We detail the sources of our improvement through an ablation study on the *base* version of our model. The ablation was done using the base version instead of

[3]Adding GNNs on top of Transformers is orthogonal to our work, and can still be added to close the gap between our model and graph-based models.

the large version of RoBERTa because of computation and time constraints. The base version fits into our smaller, more available GPUs, while the large version needs a longer time on our limited number of larger GPUs.

3.6 Discussion

3.6.1 Reader Module

In Table 3.3, we compare the performance of our proposed reader module against different readers as described in Section 3.4.2. Using predicted SFs in "SFReader" and "FFReader" hurts the performance compared to "UnfocusedReader" because the accuracy of the SFF module is not optimal. When we use gold SFs (optimal SFF), we see clear improvement over the "UnfocusedReader" baseline. When joint training "FFReader" with the SFF module, we see that it alleviates the inaccuracy of SFs. We hypothesize that the improvement comes from SFs being dynamic, and not treated as sub-optimal gold annotations. Joint training did not give any noticeable improvement when applied on the "UnfocusedReader," likely because it does not use any SFs, thus independent from the SFF module.

When joint training, even though the gradients pass through SFF and Reader modules separately, the amount of loss is what is impacted. When the Reader makes a wrong prediction based on wrong SFs, it is penalized more than if the predicted SFs were correct. This adjusts the Reader's dependence/confidence on the predicted SFs. In Table 3.3, we see that if gold SFs are used, there is no need for joint training.

3.6.2 BET with SFF Module

In Table 3.4, we show the effect of BET on our SFF module. We also evaluate on gold paragraphs to see the upper bound of the SFs prediction, while unaffected by the accuracy of selecting correct paragraphs. The difference between gold and non-gold evaluation is smaller with BET (2.58/1.91 vs 1.82/1.36), suggesting that the proposed BET not only improves the SFs predictions but also paragraph selection. We think BET is an important signal because of the way HotpotQA examples are

Model	No BET		BET	
	EM (B/C)	F1 (B/C)	EM (B/C)	F1 (B/C)
Unfocused	63.02 (61.47/69.22)	76.96 (77.40/75.20)	63.84 (62.93/67.43)	77.80 (78.80/73.75)
SFReader	60.85 (60.31/62.99)	74.51 (75.85/69.17)	61.78 (61.06/64.64)	75.20 (76.41/70.36)
FFReader	61.00 (60.42/63.29)	74.93 (76.41/69.02)	61.80 (61.42/63.29)	75.79 (77.47/69.17)
+ joint train	63.62 (62.05/69.87)	77.73 (78.19/75.89)	**64.80** (63.60/69.62)	**78.46** (79.18/75.59)
+ gold SFs	65.55 (64.40/70.17)	79.36 (80.32/75.49)	66.71 (65.90/69.92)	80.14 (81.19/75.94)

Table 3.3: Comparison between dev scores for "Ans" task of different types of *base* reader modules as described in Section 3.4.2. We split the scores into (B/C), meaning the relative scores for (Bridge/Comparison) question types. FFReader with BET and joint training is our proposal.

Model	No BET		BET	
	EM (B/C)	F1 (B/C)	EM (B/C)	F1 (B/C)
SFF module	59.00 (55.73/72.01)	85.85 (84.81/89.99)	**61.04** (58.78/69.87)	**87.16** (86.44/89.99)
+ joint train	60.46 (57.73/71.26)	86.87 (85.89/90.78)	60.20 (57.67/70.27)	87.08 (86.36/89.94)
+ gold para	61.58 (58.82/72.56)	87.76 (87.10/90.38)	62.86 (60.85/70.86)	88.52 (88.04/90.43)

Table 3.4: We show the effect of BET and joint training on the *base* SFF module's dev scores for the "Sup" task. The use of gold paragraphs (excluding paragraph pair selection) shows the upper bound of SFs prediction. We split the scores into (B/C) meaning the relative scores for (Bridge/Comparison) question types.

collected; "Bridge" type questions, which are the majority, are anchored around the bridge entity connecting the paragraphs.

3.6.3 Effect of Joint Training on SFF Module

We see that joint training slightly harms the performance of the SFF module if BET is used, while improves it without BET. We think that it might be because even when a bridge entity is present in the training instances, there is no guarantee that there is an answer (the instances other than 2 gold paragraphs as explained in Section 3.2.2), which may give conflicting signals for the BET existence. Even with this negative effect, the benefit of using BET outweighs the benefit of joint training the SFF module without BET.

3.6.4 Performance Per Question Type

We separate the results in Tables 3.3 and 3.4 by question type, and we indeed see that BET benefits the "Bridge" question types much more than the "Comparison" types. In some cases, it slightly harms "Comparison" types, likely because there is usually no hop between comparison paragraphs, so bridge entities act as distractions. As one possible future work, we can add a classifier to predict the question type and only tag bridge entities when the question is not a "Comparison" type.

Compared to the jointly trained FFReader, we see from Table 3.3 that there is almost no difference in the score of the "Comparison" question types when using gold SFs. This suggests that the flexible focusing on SFs is most important in "Bridge" question types.

3.6.5 Paragraph Pair Scoring

In addition to the method in Eq. 3.4, we experiment with two other alternative paragraph pair scoring methods as follows:

$$Pair_i = \sum_{j \in S_i} P_j \tag{3.6}$$

$$Pair_i = -no\_ans^i_{logit} \tag{3.7}$$

Scoring method	Ans	Sup	Joint
	EM / F1	EM / F1	EM / F1
$-no\_ans^i_{logit}$	63.32 / 76.88	46.08 / 79.23	31.88 / 62.96
$\sum_{i \in S_i} P_i$	64.55 / 78.23	61.04 / 87.16	42.42 / 69.81
$\sum_{i \in S_i} P_i - no\_ans^i_{logit}$	**64.80 / 78.46**	**61.19 / 87.39**	**42.48 / 70.08**

Table 3.5: Comparison between pair scoring methods in evaluation time using the *base* version of our final model (Jointly trained FFReader module + separately trained SFF module + BET).

where we either only use paragraph classification scores from the SFF module, or we only use the *no_answer* score from the reader module. We show a comparison between the three methods in Table 3.5. We notice that the paragraph classification score in the SFF module is more important than no_ans_{logit} from the reader module, but their combination gives the best paragraph pair classification accuracy.

3.6.6 Initialization of Joint Training Experiments

All joint training experiments presented in this paper are *fresh* runs, meaning that the parameters of SFF and reader modules were initialized randomly. We also experiment with another type of training where those two modules were initialized with weights of separately trained SFF and reader modules. We fine-tune the initialized modules for fewer epochs and several learning rates, but we find that such training barely improves the reader. We compare the best-performing trial against separate and fresh joint training in Table 3.6.

As mentioned in Section 3.6.1, FFReader suffers without joint training because it is trained with predicted SFs which are sub-optimal (Compared to the results in Table 3.3 where it is trained with gold SFs). Now fresh joint training alleviates this by using dynamic SFs, and also, the loss of the SFs misprediction helps the reader module become less dependent on the SFs when necessary (when they are wrong). The reader module in joint fine-tuning is initialized by a reader that was trained with sub-optimal SFs, and it only has limited training to fix its confidence

Score of	Separate training	Fresh joint training	Joint fine-tuning
	EM / F1	EM / F1	EM / F1
SFF	**62.86** / 88.52	61.58 / 88.30	62.53 / **88.56**
FFReader	62.57 / 76.40	**65.39** / **79.01**	62.94 / 76.77
Joint	- / -	**42.48** / **71.07**	42.16 / 69.40

Table 3.6: Comparing joint fine-tuning with fresh joint training and separate training. We use BET with the *base* version of the model, and we evaluate only on gold paragraphs.

Data split	Examples Count	> 512	> 768	> 1,024
Training with neg.	270,817	19,693	1,830	371
Development	7,405	164	5	0

Table 3.7: Length statistics about the input of different data splits. Training with neg. means adding negative samples of paragraph pairs to the 2 gold paragraph pair as described in Section 3.2.2. We use this split in our actual training.

in the sub-optimal SFs. We see that in the small improvement in Table 3.6 where reader performance only improves about 0.35 EM/F1 scores.

3.6.7 Plain RoBERTa versus LongRoBERTa

To justify the use of 1,024 tokens instead of 512 of a plain RoBERTa, we show statistics of input lengths in Table 3.7. We see that 7% of our training instances (Training with neg.) and 2.2% of gold paragraph pairs in the dev split go over the 512 limit. To further study the effect of truncating those examples, we train our model using a plain RoBERTa instead of a LongRoBERTa as the basic encoding unit and show the results in Table 3.8. We find that the performance drops about 1.0/0.7 EM/F1 for the reader module, which shows the benefit of using longer sequences. In Table 3.2, models that outperform our model while only using 512 tokens also use GNNs, which explains their performance boost.

Score of	Plain RoBERTa	LongRoBERTa
	EM / F1	EM / F1
SFF	**62.5** / **88.89**	61.58 / 88.30
FFReader	64.32 / 78.31	**65.39** / **79.01**
Joint	**42.80** / 70.85	42.48 / **71.07**

Table 3.8: A comparison between using our model with plain RoBERTa versus LongRoBERTa.

3.6.8 Joint Training Hyperparameter λ

We experiment with different λ values in Eq. 3.5 and we show the details in Table 3.9. We see that for all values, SFF performs worse than the separately trained SFF module, thus we opt for the λ value that most improves our reader module, which is 0.5.

3.6.9 Case Study

In Table 3.11, we show some examples that are improved by our FFReader and BET proposals. We show links that are tagged with [BE][/BE] as underlined, wrong answers in {brackets}, correct answers in **bold**, and supporting facts in *italic*.

Questions 1 and 2 demonstrate the effectiveness of our FFReader. In Question 1, Unfocused Reader tries to find a time span that can be the answer to the question from both paragraphs with no guidance on what sentences are important, while FFReader used the tagged SFs and showed how focusing on SFs helps to find the answer. Question 2 shows how our model can still predict answers outside the predicted SFs, while SFReader models like TAP2 and QUARK are limited to answers within the SFs. In this example, both SFReader and FFReader have the same predicted SFs, but SFReader can only see the SFs, so the only driver name it can give is "Sergio Pérez," while FFReader considers the SFs but chooses the correct answer outside of them.

Questions 3 and 4 demonstrate the effect of our BET proposal. In question 3, we see that the Unfocused Reader without BET gives a correct answer type (a

Score of	Focus on Reader λ=0.1	0.25	0.5	0.75	Focus on SFF 0.9
	EM / F1	EM / F1	EM / F1	EM / F1	EM / F1
SFF module	61.92 / **88.55**	61.81 / 88.26	61.58 / 88.30	61.84 / 88.48	**62.07** / 88.52
FFReader module	65.04 / 78.62	64.90 / 78.54	**65.39** / **79.01**	64.59 / 78.57	64.50 / 78.20
Joint evaluation	**42.80** / 70.59	42.57 / 70.59	42.48 / **71.07**	42.39 / 70.72	42.62 / 70.47

Table 3.9: Dev scores of joint *base* experiments on the distractor setting of HotpotQA with varying λ of Eq. 3.5. All experiments use BET.

language), but the wrong answer. Without using BET, the system did not consider the importance of the second paragraph, and it just guessed one language near the word "Padosan." However, with the bridge clearly marking the importance of the linked document, the system was able to find the correct answer. In Question 4, FFReader without BET chose "Tunisian" as the answer because it appeared before "historian" which matches the question, without considering the other paragraph. With BET, the system paid more attention to the related paragraph and found the correct answer.

3.6.10 Problems in HotpotQA Annotations

In this section, we show that another reason why robustness in dealing with SFs is important is because annotations in HotpotQA can sometimes be inconclusive; they do not actually cover all the required sentences for reasoning. In Table 3.10, we show an example of HotpotQA SFs annotation issue that we think hurts the training of the SFF module. We see that the sentence S_{2_1} is not considered a gold SF, even though it includes the name of one of the entities in the question. If the meaning of "supporting facts" is that they are the only sentences required for reasoning to arrive at the answer, then if the reader has access *only* to these sentences, there is no way to resolve the pronoun "He" in sentence S_{2_1}. This would be considered an annotation mistake, and we encountered many such annotations.

If the meaning of "supporting facts" is that they are the core sentences required for making the final reasoning decision (not necessarily including all pronoun resolutions), then there is a logical annotation mistake, because the sentence that contains the answer is always a supporting fact, while other sentences that could have been the answer are not. An example of why this hurts the performance is as follows: S_{1_1} and S_{2_1} are equally important; they just define the players. If our SFF module predicts S_{2_1} as an SF, it would get penalized, even though this is a totally logical prediction. In order to have the SFF module achieve 100% EM accuracy, it would need to know the answer before predicting the SFs. After sampling 20 examples, we found 5 examples with this problem, which means around 25% of questions have this issue.

Question:	Which tennis player won more Grand Slam titles, Henri Leconte or Jonathan Stark?
Answer:	Jonathan Stark
Paragraph:	Jonathan Stark
[Gold SF] S_{1_1}	Jonathan Stark (born April 3, 1971) is a former professional tennis player from the United States.
[Gold SF] S_{1_2}	During his career he won two Grand Slam doubles titles (the 1994 French Open Men's Doubles and the 1995 Wimbledon Championships Mixed Doubles).
Paragraph:	Henri Leconte
S_{2_1}	Henri Leconte (born 4 July 1963) is a former French professional tennis player.
[Gold SF] S_{2_2}	He reached the men's singles final at the French Open in 1988, won the French Open men's doubles title in 1984, and helped France win the Davis Cup in 1991.

Table 3.10: Example from HotpotQA where SFs annotations are inconclusive.

Question 1:	When was the Western Germanic language spoken from which the small settlement situated on the river Leda opposite Leer derives its name?
Gold Answer:	between the 8th and 16th centuries
Paragraph:	Leda (river) *[SF] The Leda is a river in north-western Germany in the state of Lower Saxony. [/SF]* It is a right tributary of the Ems and originates at the confluence of the Sagter Ems and the Soeste (Dreyschloot) near the town of Barßel. The Leda flows into the Ems near the town of Leer.
[Gold SF]	*[SF] On the southern bank of the Leda, in the "Overledingen Land" (Overledingen="country over the Leda"), opposite Leer, lies the small settlement of Kloster Muhde ("Muhde" from the Old Frisian "mutha" meaning "(river) mouth") [/SF].* The total length of the river is 29 km , of which the lower 1.9 km are navigable for sea-going vessels.
Paragraph:	Old Frisian
[Gold SF]	*[SF] Old Frisian is a West Germanic language spoken* ***between the 8th and 16th centuries*** *in the area between the Rhine and Weser on the European North Sea coast. [/SF]* The Frisian settlers on the coast of South Jutland (today's Northern Friesland) also spoke Old Frisian but no medieval texts of this area are known. The language of the earlier inhabitants of the region between the Zuiderzee and Ems River (the Frisians mentioned by Tacitus) is attested in only a few personal names and place-names. Old Frisian evolved into Middle Frisian, spoken from the {16th to the 19th century}.

Answer of UnfocusedReader with BET: 16th to the 19th century

([SF] tags were not used in UnfocusedReader)

Answer of FFReader with BET: **between the 8th and 16th centuries**

Question 2:	Which other Mexican formula one race car driver has held the podium besides the Force India driver born in 1990?
Gold Answer:	Pedro Rodríguez
Paragraph:	Forumula One drivers from Mexico There have been six Formula One drivers from Mexico who have taken part in races since the championship began in 1950.
[Gold SF]	**Pedro Rodríguez** is the most successful Mexican driver being the only one to have won a Grand Prix.
[Gold SF]	*[SF] Sergio Pérez, the only other Mexican to finish on the podium, currently races with Sahara Force India F1 team. [/SF]*
Paragraph:	Sergio Pérez
[Gold SF]	*[SF] {Sergio Pérez} Mendoza (; born 26 January 1990) also known as "Checo" Pérez, is a Mexican racing driver, currently driving for Force India. [/SF]*

Answer of SFReader with BET: Sergio Pérez

Answer of FFReader with BET: **Pedro Rodríguez**

Question 3:	Padosan had a supporting actor who is known as a successful playback singer in what language?
Gold Answer:	Hindi
Paragraph:	Padosan Padosan (Hindi: "", {English}: lady Neighbour) is a 1968 Indian comedy film. Directed by Jyoti Swaroop. It was produced by Mehmood, N. C. Sippy and written by Rajendra Krishan. It was a remake of the Bengali film "Pasher Bari" (1952) starring Bhanu Bandyopadhyay and Sabitri Chatterjee. The movie stars Sunil Dutt and Saira Banu in lead roles.
[Gold SF]	Kishore Kumar, Mukri, Raj Kishore and Keshto Mukherjee played the supporting roles.

	Mehmood as the South Indian musician and rival to Sunil Dutt is among the highlights of the film. It was considered as one of the best comedy movies ever made in Hindi film history. Mehmood's portrayal of a south Indian music teacher was one of his all time best and noted performances and a key highlight of the film. Kishore Kumar's character of a comical theater director was also well received. "Indiatimes Movies" ranked the movie amongst the "Top 25 Must See Bollywood Films". Music was composed by R.D. Burman and was a huge hit. Kishore Kumar sang for himself while Manna Dey sang for Mehmood.
Paragraph:	Kishore Kumar Kishore Kumar (4 August 1929 – 13 October 1987) was an Indian playback singer, actor, lyricist, composer, producer, director, and screenwriter.
[Gold SF]	He is considered one of the successful playback singers in the **Hindi** film industry.
Answer of UnfocusedReader without BET: English	
Answer of UnfocusedReader with BET: **Hindi**	
Question 4:	Georges-Henri Bousquet translated the work of a historian who is of what heritage?
Gold Answer:	North African Arab
Paragraph:	Georges-Henri Bousquet Georges-Henri Bousquet (21 June 1900, Meudon – 23 January 1978, Latresne) was a 20th-century French jurist, economist and Islamologist. He was Professor of law at the Faculty of Law of the University of Algiers where he was a specialist in the sociology of North Africa (Berbers, Islam).

[Gold SF]	He is also known for his translation work of the great Muslim authors, Al-Ghazali, a theologian who died in 1111 and {Tunisian} historian Ibn Khaldun (1332-1406). He was known as a polyglot, spoke several European languages (Dutch, his second mother tongue, English, German, Italian, but also Spanish, Danish, Norwegian ...) and Eastern ones (Arab, Malay ...).
Paragraph:	Ibn Khaldun
[Gold SF]	Ibn Khaldun (; Arabic: , "Abū Zayd ʿAbd ar-Rahman ibn Muhammad ibn Khaldūn al-Hadrami" ; 27 May 1332 – 17 March 1406) was a **North African Arab** historiographer and historian.
Answer of FFReader without BET: Tunisian	
Answer of FFReader with BET: **North African Arab**	

Table 3.11: Examples from the development split of HotpotQA distractor setting. We compare the results of several systems that are shown in Table 3.3 as follows: Question 1) UnfocusedReader v.s. FFReader. Question 2) SFReader v.s. FFReader. Question 3) UnfocusedReader with and without BET. Question 4) FFReader with and without BET. All FFReader models were trained with joint training while using a separately trained SFF module, as explained in Section 3.4.4.

3.7 Summary of This Chapter

In this chapter, we propose a multi-hop QA model that: 1) Uses supporting facts to answer questions in a novel way, 2) Tags bridge entities that connect paragraph pairs, and 3) Jointly train separate modules for answer, and supporting facts prediction. Our model outperforms all non-graph-based models in answer finding and achieves comparable scores with state-of-the-art graph-based models. For future work, we want to explore applying global attention to entities to explore if it can mimic the GNNs that are applied to entities.

Chapter 4

Investigating the Factual Knowledge Memory in Closed-Book Setting

Recent research shows that Transformer-based language models (LMs) store considerable factual knowledge from the unstructured text datasets on which they are pre-trained. The existence and amount of such knowledge have been investigated by probing pre-trained Transformers to answer questions without accessing any external context or knowledge (also called closed-book question answering (QA)). However, this factual knowledge is spread over the parameters inexplicably. The parts of the model most responsible for finding an answer only from a question are unclear. This study aims to understand which parts are responsible for the Transformer-based T5 reaching an answer in a closed-book QA setting. Furthermore, we introduce a head importance scoring method and compare it with other methods on three datasets. We investigate important parts by looking inside the attention heads in a novel manner. We also investigate why some heads are more critical than others and suggest a good identification approach. We demonstrate that some model parts are more important than others in retaining knowledge through a series of pruning experiments. We also investigate the roles of encoder and decoder in a closed-book setting.

Neural language models (LMs) store knowledge from the unstructured text on which they are pre-trained (Roberts et al., 2020; Petroni et al., 2019; Jiang et al., 2020). Such a phenomenon of storing knowledge is interesting and valuable because LMs are trained on unstructured text and are easy to query. Previous work investigated the existence and amount of such knowledge in Transformer-based models, such as BERT (Devlin et al., 2019) and T5 (Raffel et al., 2020), and how to effectively query models to find the stored knowledge.

Querying these models is a form of question answering (QA), and it is considered a direct interface to the knowledge stored in LMs. Given that using context to answer questions makes it difficult to know if the model has retrieved this knowledge from within or from the context, we focus on analyzing this phenomenon using closed-book (QA) (Section 4.1). This setting is used by Roberts et al. (2020) to feed T5 models open-domain questions with no context to measure the amount of knowledge stored. In open-book QA, models are either fed context that contains the answer (Rajpurkar et al., 2016) or allowed to use external knowledge sources (Chen et al., 2017). However, in closed-book QA, the model only depends on the question and "look up information" stored in its parameters to find the answer. Therefore, closed-book QA is essential to estimate the amount of stored knowledge. For example, only feeding "What is the capital of Japan?" is sufficient for the model to generate the answer.

Although the amount of knowledge stored after pre-training is estimated using closed-book QA (Roberts et al., 2020), how this knowledge is stored and which parts of the T5 model are responsible for reaching answers remains poorly understood. This study aims to understand how knowledge is stored inside T5 in a closed-book QA setting while also studying the importance and role of different parts of the model (Section 4.2). We also present head importance scoring methods for pruning the model to verify the distribution of knowledge (Section 4.3). Understanding the inner workings and the crucial parts of the model allows for several benefits, such as guided pruning, where the size of LMs is reduced while preserving learned knowledge, aiding better design choices, and providing hints about how to integrate external knowledge with LMs better.

The contributions of this study are three-fold:

- After a novel investigation of the components of attention, we find that the query component is the most important, as it is almost the only component that changes when back-propagation is calculated.
- We find that the encoder finds the embedding space of the desired answer entity, and the decoder generates the textual representation of that entity.
- We compare different attention-head importance scoring methods on three datasets and find that the stored knowledge is not evenly distributed in the model.

4.1 Preliminary

4.1.1 Task

This study aims to understand how QA Transformer-based T5 models generate answers to factual questions in a closed-book setting. It also attempts to find whether parts of the model are more important than others in reaching answers and then identify them.

In a closed-book QA setting, the model is given a question and is expected to provide an answer without accessing context or external knowledge. The model must "memorize" the answers given during the pre-training or fine-tuning. How Transformers work in this setting is principally different than other tasks. In other settings, the model generally uses attention to focus on some parts of the provided context. For instance, the model can extract answers from a context paragraph in a normal QA setting (Devlin et al., 2019) or align words between source and target languages as in translation tasks. However, because only the question is provided in a closed-book QA setting, the model can only focus on the question tokens, and how it decides the answer is unclear.

4.1.2 Datasets

Following Roberts et al. (2020), we use three open-domain QA datasets: Natural Questions (NQ) (Kwiatkowski et al., 2019), WebQuestions (WQ) (Berant et al., 2013b), and TriviaQA (TQA) (Joshi et al., 2017). NQ and WQ are both datasets

of questions from web queries, while TQA is a collection of questions from quiz league websites. All questions are accompanied by context of different forms to help find the answer. However, to use the datasets as closed-book QA datasets, all context is ignored, and only the questions are considered.

Our analysis uses the T5 (Raffel et al., 2020) models provided by Roberts et al. (2020). In particular, we use three variants of T5: T5-Large, T5-11b, and T5v1.1-xxl. Details about them are explained in Section 4.3.3.

4.1.3 Transformer Architecture

We briefly describe the Transformer architecture (Vaswani et al., 2017), which comprises the T5 model while focusing on the attention mechanism therein.

The Transformer is a model architecture with blocks of two stacked sub-layers: multi-head self-attention and feed-forward (FF) networks. Given an input sequence $\mathbf{x} = \mathbf{x}_1, ..., \mathbf{x}_n \in \mathbb{R}^{d_{model}}$, for each attention head h, a query Q^h, a key K^h, and a value V^h are computed from the input $\mathbf{x}$ as follows:

$$\begin{aligned} Q^h &= \mathbf{x}W_q^h \\ K^h &= \mathbf{x}W_k^h \\ V^h &= \mathbf{x}W_v^h \end{aligned} \tag{4.1}$$

where W_q^h, W_k^h, and W_v^h are parameter matrices for the head h. Q^h, K^h and V^h matrices contain $\mathbf{q}^h$, $\mathbf{k}^h$, and $\mathbf{v}^h$ vectors of different input tokens. Then, the attention scores α^h are computed as:

$$\alpha^h = \text{softmax}(\frac{Q^h {K^h}^T}{\sqrt{d_k}}) \tag{4.2}$$

where d_k is the key dimensionality. Then, the contextualized Z^h is computed by scaling V^h by attention scores as follows:

$$Z^h = \alpha^h V^h \tag{4.3}$$

with Z^h containing $\mathbf{z}^h$ vectors of the input tokens. The heads are then combined to produce Z as follows:

$$Z = \text{concat}_{h=1}^{H}(Z^h)W_o \tag{4.4}$$

where $\text{concat}_{h=1}^{H}$ is a function that concatenates Z^h of all H heads and W_o is a parameter matrix.

Next, the group of attended vectors Z is passed to the next component, a FF network. Vaswani et al. (2017) proposed using a two-layer network with ReLU activation. The final model has N layers of such blocks. When using the block in the decoder instead of the encoder, an additional sub-layer of cross-attention is added after the self-attention. This additional sub-layer receives its K and V from the output of the encoder.

4.2 Understanding Transformer-Based T5 in Closed-Book QA

The main component of Transformers is the attention mechanism; therefore, we start by looking into its components. Then, we verify the findings and finally investigate the role of the encoder/decoder as a whole.

4.2.1 Dissecting Attention Components

Initially, we look at the gradients (or error back-propagation) resulting from the mismatch between the model prediction and the golden answer. They represent where the model attempts to fix itself to predict the correct answer. Areas with high gradients represent important parts responsible for reaching the answer. We gather these gradients and attempt to find interesting patterns.

Gathering Gradients

We evaluate the model on the validation set of NQ while calculating loss and applying a backward pass to obtain the gradients *without updating the parameters*. The loss is calculated against the gold labels for all tokens in one go using the built-in loss function in the T5 implementation of the Transformers library (Wolf et al., 2020). We reduce the calculated gradients for each example (consisting

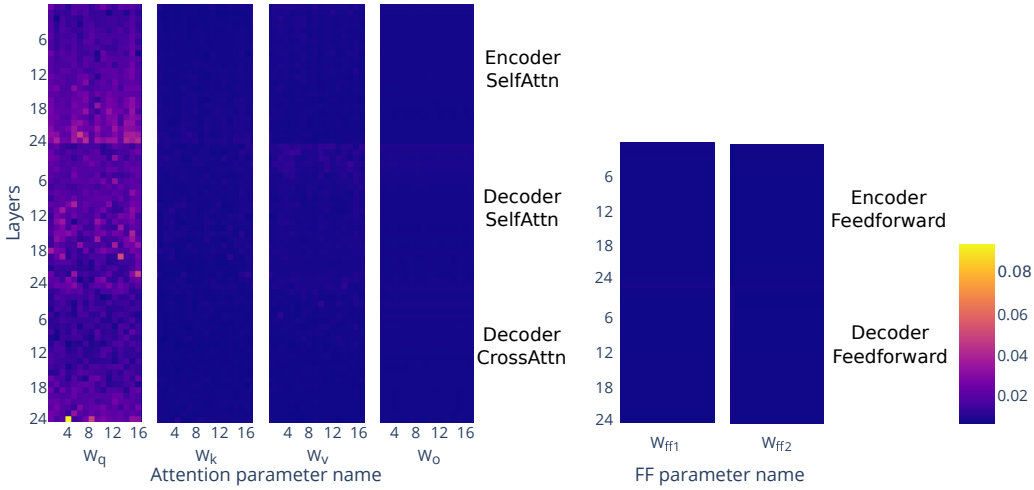

Figure 4.1: Visualization of the averaged gradients of the attention and FF parameters for T5-Large over the validation split of the NQ dataset.

of a question and a golden answer) by averaging each parameter matrix. The attention parameter matrices reduced are:

$$W_q^h, W_k^h, W_v^h \in \mathbb{R}^{d_k \times d_{model}}$$
$$W_o \in \mathbb{R}^{H \cdot d_k \times d_{model}}$$

where W_q^h, W_k^h, W_v^h and W_o are from the attention sub-layer as explained in Section 4.1.3. For completeness, we also analyze the FF matrices:

$$W_{ff1}^T, W_{ff2} \in \mathbb{R}^{ff_{size} \times d_{model}}$$

where W_{ff1} and W_{ff2} are the learnable matrices of the two-layer FF networks of each block. We then average the reduced results from all the examples.

Visualizing Gradients

We use the T5-Large provided by Roberts et al. (2020) with the NQ dataset to gather gradients, as explained previously. Figure 4.1 visualizes the reduced gradients of the three attention types: Encoder.SelfAttention, Decoder.SelfAttention and Decoder.CrossAttention (shortened to EncSA, DecSA, and DecXA, respectively), each having 24 layers. Each row represents a layer and consists of the

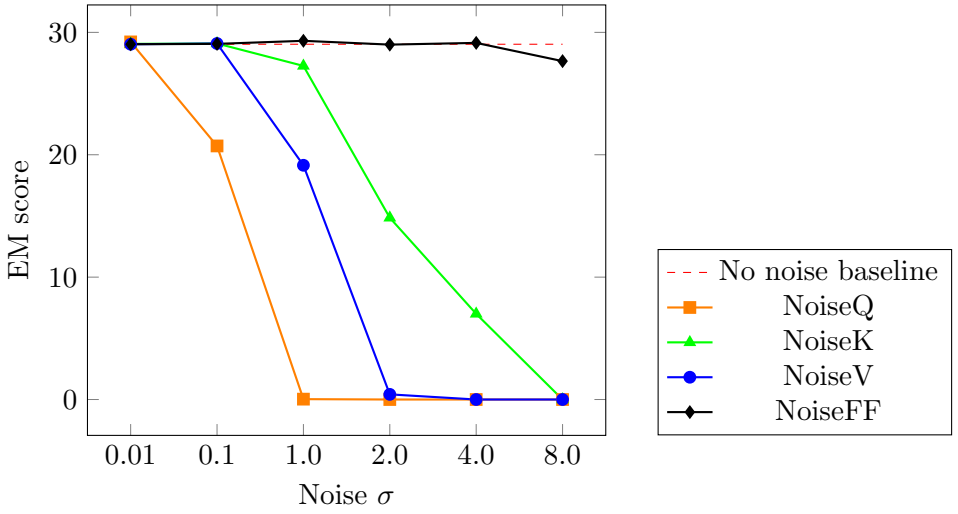

Figure 4.2: Evaluation of T5-large on NQ dataset with noise of magnitude λ added to different attention components and FF output.

16 heads of T5-Large. We also visualize the FF parameters of the encoder and decoder.

Interestingly, as shown in Figure 4.1, almost only W_q is changed in all attention types. While later layers are changed more in the EncSA and DecXA, the middle layers have the most change for DecSA. We notice minor changes in W_v from Figure 4.1 and almost no change in the FF network parameters. This implies that, when the model predicts a wrong answer and the gradients are applied to fix the prediction, almost only **q** vectors are changed. We apply a similar analysis to other variants of T5 (T5-11b and T5v1.1-xxl) and find similar results.

Confirming the Importance of Q

Having shown the importance of Q for correcting wrong predictions, we conduct another experiment to confirm the importance and the applicability to correctly predicted answers. We add a normal noise $N(0.0, \sigma)$ to **q**, **k**, **v**, or FF vectors and observe the performance change, where σ is the standard deviation. A larger σ represents a more substantial noise.

We evaluate T5-large on the NQ dataset with noise added to one attention

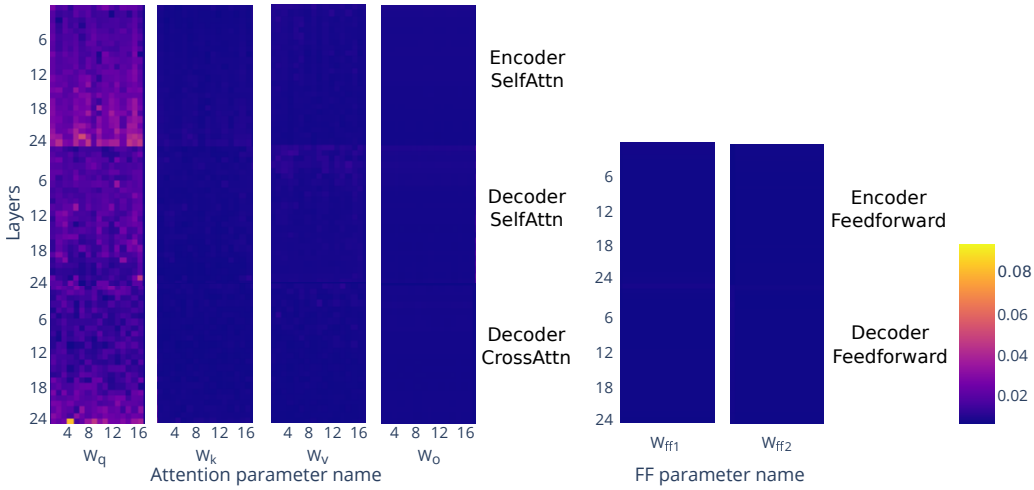

Figure 4.3: Visualization of the averaged gradients of the attention parameters for T5-Large over the validation split of the NQ dataset after shuffling all target answers to random answers of a different named entity type.

component or FF at a time. We show the results in Figure 4.2 and compare them to the baseline of T5-large with no noise, which is 29.03 exact match (EM). Applying noise to **q** vectors degrades the performance substantially faster than other components, affirming the importance of the query component for wrongly and correctly predicted answers. The FF networks were most resilient to noise because of the drop-out training.

We assume the following: (1) The behavior of not changing W_k, W_v, or W_o is related to the fact that changes required to fix predictions are typically small. (2) A drastic change in the expected answer invokes more changes in, for example, W_v. These assumptions can be answered by considering the type of predictions and golden answers.

We use SpaCy (Honnibal et al., 2020) to apply named entity recognition (NER) on all answers and compare the types of pairs of {prediction, gold} answers. The majority are of the same type (e.g., (GPE, GPE) or (DATE, DATE)). We then shuffle the golden answers, replacing each with a random golden answer of a different type. This would introduce many type mismatches, and the model would require bigger gradients/changes to fix the predictions. Then, we evaluate

and gather the gradients again, as in Section 4.2.1 and show them in Figure 4.3. Interestingly, the only change between Figs. 4.1 and 4.3 was slightly brighter W_q, with almost no change to the other attention parameters.

To verify whether W_k and W_v are changed during fine-tuning, we fine-tune a raw version of T5 on the NQ dataset and apply the gradient analysis explained earlier. While still only W_q was highly changed, at 50 and 250 steps, the decoder had significantly more gradients than the encoder, whereas, at 1k and 2k, the encoder had higher gradients. The results in Figure 4.1 are after fine-tuning ends. The lack of change in W_k and W_v during fine-tuning implies that they were changed (and probably learned optimally) during the pre-training phase.

Difference Between Q and K

Although Q^h and K^h are computed by the same input and formula (except the weights), only W_q receives high gradients. This gradient difference comes from the matrix multiplication order. To calculate the weight a^h, we do $Q^h {K^h}^T$, where the resulting matrix has rows that correspond to the queries and the columns to keys. Then, we take softmax on the last dimension, that is, by row. The softmax lets the row sum to 1.0, which would be the weights sum of our current query word attending to all other key words. The model learns this distinction between queries and keys and learns to change the queries rather than the keys.

How Knowledge Is Stored Inside T5

As experiments (Figs. 4.1 and 4.2) show that **q** vectors are the most crucial component for fixing predictions or reaching answers, we conclude that the following is how T5 stores knowledge.

In Eqs. 4.2 and 4.3, the role of **q** vectors is only to calculate a score that represents how much **v** to take from each token. Multiplying a scaling score with a vector does not change its direction, but only the norm of that vector. The **z** vectors are only composed from combining differently weighted **v** vectors (Eq. 4.3). For easier imagination, **v** vectors can be considered pointers or arrows of different lengths that point somewhere in the embedding space. When **z** vectors are combined cumulatively (over layers), they point to a place in the embedding

Prune Encoder.SelfAttention	Prune Decoder.SelfAttention
Southern Egypt	Jason Flemyng
Panama	Jason Fle
Sheryl Crow	Century City, Los Angeles
Elton John	Century City, California
Dob Robertson	a victim's dilemma
Robert Kelly	a victim of a distress signal
Subway	John Kramer
Eleven Madison Park	John C. J. Miller

Table 4.1: Patterns of the differences in predictions when only pruning EncSA vs DecSA. Each row's first and second lines show the prediction before and after pruning.

space that represents the desired answer. In summary, Q^h is a map that guides the V^h arrows/pointers to this final point in the embedding space.

4.2.2 Role of Encoder/Decoder

Until now, we have discussed attention head components as parts of the encoder–decoder model. Encoders and decoders usually have straightforward roles in other tasks, such as focusing on source/target languages and performing alignment in translation tasks, or selecting text spans in open-book QA. However, in a closed-book QA setting, their role is unclear. To investigate their role in a closed-book QA setting, we remove attention heads from one attention type at a time (e.g., EncSA) or FFs from the encoder or decoder and check the differences in predictions before and after a significant drop in performance. The procedure for removing heads is introduced in Section 4.3. We apply the same procedure for removing FFs. Comparing the results by removing EncSA heads and DecSA heads presented in Table 4.1, the role of the encoder is finding the correct entity while the decoder generates the textual representation of that entity. We did not witness special patterns when removing heads only from DecXA.

Table 4.2 shows a quantitative comparison of how predictions change before

	EM	Textual Overlap	No Textual Overlap
Only pruning heads in encoder	36.70%	10.39%	52.91%
Only pruning FFs in encoder	43.32%	11.16%	45.51%
Only pruning heads in decoder	54.49%	23.74%	21.77%
Only pruning FFs in decoder	46.15%	25.51%	28.34%

Table 4.2: Quantitative comparison between the predictions before and after substantial pruning (not accuracy against gold answers).

and after substantial pruning (with about 10 EM points drop). All predictions after pruning can either match predictions before pruning (EM), differ but with textual overlap, or become different with no overlap. Pruning the decoder heads does not change many predictions (high EM with predictions before pruning). It produces more wrong but textually overlapping predictions because the entity space has been decided in the encoder. However, harming the encoder heads results in predicting about 30% more non-textually overlapping predictions (i.e., different entities). Pruning the FFs in the encoder and decoder has a more negligible effect than pruning heads, but still follows the same trend.

4.3 Head Importance for Pruning

In a closed-book QA setting, the model must generate the correct answer with only access to the question tokens. The "knowledge" about the correct answer is within the model's parameters. We hypothesize that such "knowledge" is not distributed equally in the model, but there are areas that are significantly more important than others in retaining that knowledge.

To verify that some heads of the model are more important than others in retaining knowledge, we perform a series of pruning tests on attention heads according to different importance scoring methods. We explore pruning least-important heads first or most-important heads first. The former provides a practical advantage where pruning models and reducing their parameter count while preserving their knowledge allows for faster and lighter models. Because FF networks do

not store factual knowledge, as discussed in Section 4.2.1, we exclude them from further pruning experiments.

We explain the pruning settings and methods, visualize different importance maps, and compare them via pruning experiments on different datasets.

4.3.1 Pruning Flow

An importance score is given for each attention head. Then, the least/most 10% important heads are masked iteratively by adding a mask variable ξ^h following Michel et al. (2019), changing Eq. 4.3 as follows:

$$Z^h = \xi^h \alpha^h V^h \tag{4.5}$$

where the values of ξ^h are either 0 or 1. After each masking round, the model is re-evaluated, and a new head importance map is calculated for the next masking round until we reach a threshold.

4.3.2 Importance Scoring Methods

For each of the three attention types (EncSA, DecSA, and DecXA), a head importance map is generated using one of the following methods:

Random

Heads are given random importance scores. This represents a counter hypothesis that knowledge is distributed equally in all of the model.

Gradients of Mask Gates (*attn*)

The importance score for a head is the gradient of the mask variable ξ^h. This method, introduced by Michel et al. (2019), focuses on the value of the attention score α^h. When the attention score is high, the gradient of ξ^h is also high; therefore, this method is called *attn*.

Summed Gradients of Attention Parameters (*grad_avg*)

Looking at the analysis result in Section 4.2.1, we introduce a new method of calculating the importance score by summing the reduced gradients of the parameter matrices in attention heads. More specifically, for each head h, we calculate scores $grad^h_{avg}$ as follows:

$$grad^h_{avg} = \text{avg}(g(W^h_q)) + \text{avg}(g(W^h_k)) + \text{avg}(g(W^h_v))$$

where $g()$ is the function that obtains the gradients of a parameter matrix, and avg is a function that obtains the average of a matrix.[1] Since almost only W_q has high gradients among attention components (as shown in Figure 4.1), adding the gradients of W_k and W_v has almost no effect on the final importance map. Therefore, we can consider that the importance of an attention head is equivalent to the importance of only the query component of that head. However, in this method, we still add the gradients of all components for completeness.[2]

Norm of $\mathbf{z}^h$ Vectors (*norm*)

This method does not depend on the gradients but on the norm of $\mathbf{z}^h$ vectors. Kobayashi et al. (2020) used the norm for analyzing alignment in machine translation. We use it as a scoring method; for each head, h, the importance score $norm^h$ is calculated as follows:

$$norm^h = \sum_{t=1}^{n} ||\mathbf{z}^h_t||$$

where $\mathbf{z}^h_t$ is the contextualized vector $\mathbf{z}^h$ of token t, and $||\cdot||$ denotes the Euclidean norm.

[1] We also experimented with max for scoring; however, we excluded it because its performance was poor.

[2] We calculated the KL-divergence between importance maps with the gradients of all components, and maps of only W_q gradients. As the KL-divergence was 0.0001, confirming similar distributions, we exclude the latter maps.

Model	d_model	#heads	d_kv	d_ff
T5-Large	1,024	16	64	4,096
T5-11b	1,024	128	128	∼65k
T5v1.1-xxl	4,096	64	64	∼10k

Table 4.3: Main differences between used models. d_model is the dimension of the word embeddings, #heads is the number of heads, d_kv is the hidden dimension of the query, key, and value vectors, and d_ff is the dimension of the FF networks after the attention sub-layer.

4.3.3 Pruning Settings

Importance Sorting

- Local: To mask 10% of the heads, the importance scores are sorted per attention type *independently*. Then, the 10% are divided equally between the EncSA, DecSA, and DecXA attention types.
- Global: The maps of the three attention types are concatenated, and all the importance scores are sorted before selecting 10% to mask. Here, more heads might be masked from DecSA than EncSA, for example.

Normalization

- Layer normalization: Michel et al. (2019) normalize importance scores by layer using the ℓ_2 norm. We use this as the default.
- No layer normalization: We skip the layer normalization and denote the setting as *nln*.

Used Pre-trained Models

We use three variants of T5 released by Roberts et al. (2020)[3]: T5-large, T5-11b, and T5v1.1-xxl. They all have 24 layers but differ in several aspects, as

[3]The pre-trained models can be downloaded from: https://github.com/google-research/google-research/tree/master/t5_closed_book_qa

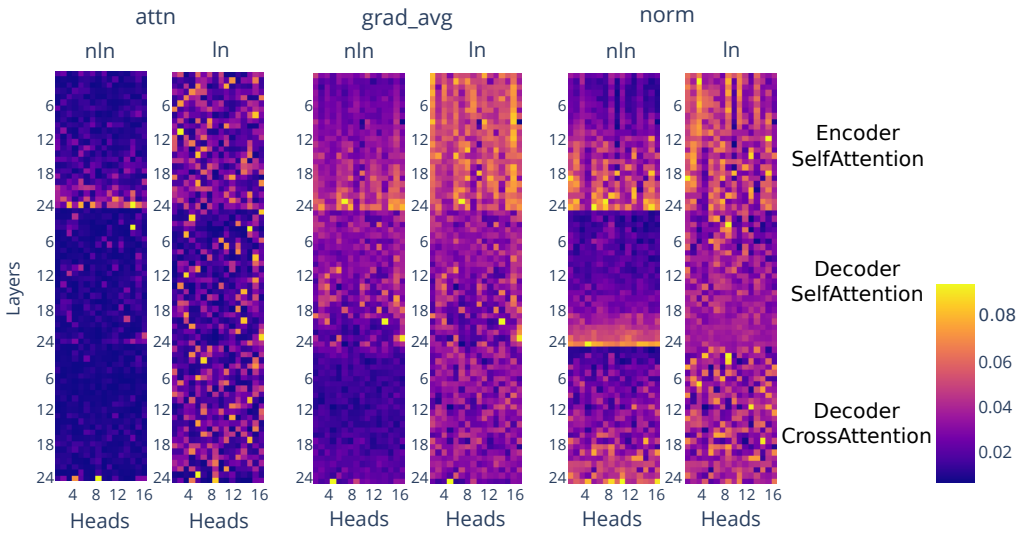

Figure 4.4: Head Importance score maps for different scoring methods with T5-large on the NQ dataset. "nln" means no layer normalization and "ln" means layer-normalized.

shown in Table 4.3. T5v1.1 also differs from other models as it has no parameter sharing between embedding and classifier layers, and it was pre-trained on C4 (Raffel et al., 2020) only without mixing in the downstream tasks. It also uses GatedGelu instead of Relu for activations. Both T5v1.1-xxl and T5-11b have around 11 billion parameters.

All models had additional pre-training with salient span masking (Guu et al., 2020) before being fine-tuned. In this objective, salient spans (named entities and dates) are mined beforehand and then used in the pre-training as masks. A fine-tuned model is released for each of the three datasets (NQ, WQ, and TQA), except T5-large, which only has a version fine-tuned on NQ.

4.3.4 Visualizing Head Importance Maps

Figure 4.4 shows the head importance maps generated with T5-large from the validation set of the NQ dataset for each scoring method. In "nln", all methods assign more importance to later layers in the EncSA but have different assessments about the DecSA. For DecXA, all methods agree on the importance of certain

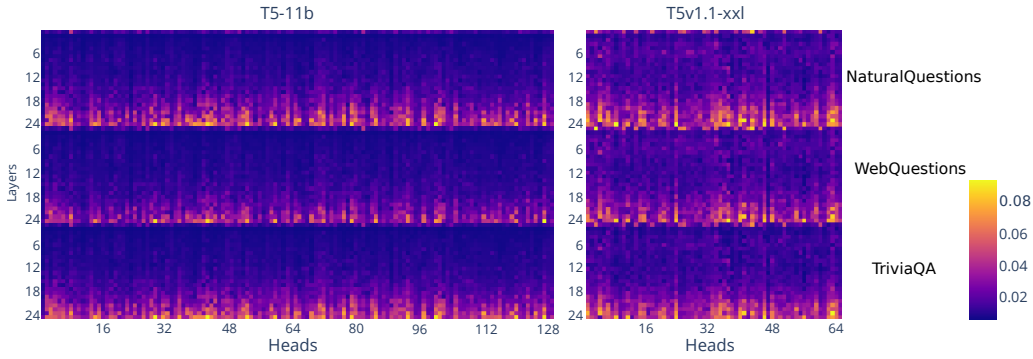

Figure 4.5: Comparing head importance visualizations of the encoders of T5-11b and T5v1.1-xxl models between different datasets, using *grad_avg* with "nln"

heads in the last layer. Although there are differences in general between methods, the importance patterns are very similar across datasets, as shown in Figure 4.5. This is likely because even though models were fine-tuned separately on the datasets, they all share the same pre-training, which is the actual source of the stored knowledge. This similarity is verified by pruning experiments in Section 4.3.5.

4.3.5 Experiments

We experiment with different scoring methods and settings with T5-large on the NQ dataset for speed. Then, we apply the best settings on the huge T5-11b and T5v1.1-xxl variants on the other datasets. We focus on "least-important heads first" in our experiments as it has a more practical benefit.

Least-Important Heads First (LIF)

We prune the least important heads first and do not expect to see a large change in performance. We stop pruning when the EM score reaches below 90% of the original score. In Figure 4.6, we show scoring methods compared to the random baseline. The global sorting setting is usually ineffective and achieves below the random baseline; therefore, we ignore it in further experiments. Layer normalization also hurts the importance scores; however, we notice different trends on huge models. We discuss the reasons later. Furthermore, since the *attn* method's per-

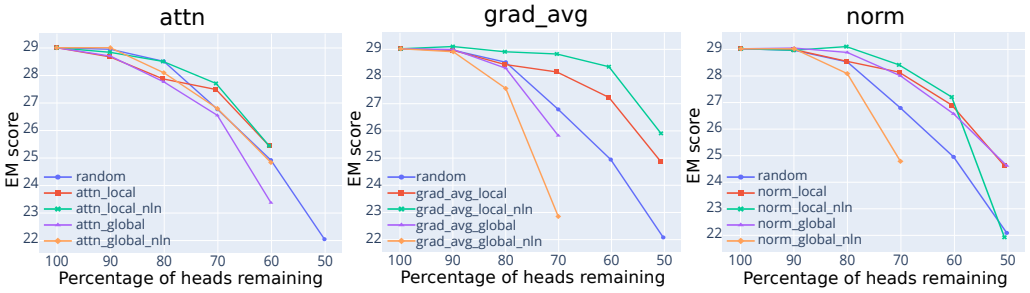

Figure 4.6: Comparing pruning experiments with different settings and methods against the random baseline. We compare *attn*, *grad_avg* and *norm* methods introduced in Section 4.3.2. In the legend, "_local" and "_global" mean local and global sorting explained in Section 4.3.3, while "_nln" means "no layer normalization" as explained in Section 4.3.3.

formance is closer to the random baseline than other methods, we exclude the *attn* method from comparison. The *attn* method showed relatively reasonable pruning performance in machine translation tasks (Michel et al., 2019) because focusing on which source/target language tokens are attended is essential. Kobayashi et al. (2020) found that only focusing on attention score ignores the norm aspect of vectors.[4]

We run the remaining settings with the huge variants (T5-11b and T5v1.1-xxl) on all datasets and show the results in Figure 4.7. While the best pruning method in T5-large is the non-layer-normalized *grad_avg* as shown in Figure 4.6, T5v1.1-xxl brings the performance of the *norm* method closer to *grad_avg*. In T5-11b, the *norm* method outperforms the non-layer-normalized *grad_avg* and slightly outperforms the *grad_avg*.

We think the reason is that the standard deviation between examples for the *norm* scores of heads is mostly lower than that for the *grad_avg* method, especially in the areas of low importance. However, small models have a more uniform importance map when normalized, as shown in Figure 4.4. Such distribution leads the pruning toward a more random selection process. In models with many

[4]Experiments of *attn* on other datasets and variants were also conducted, but because of poor performance, we exclude it from graphs for visibility.

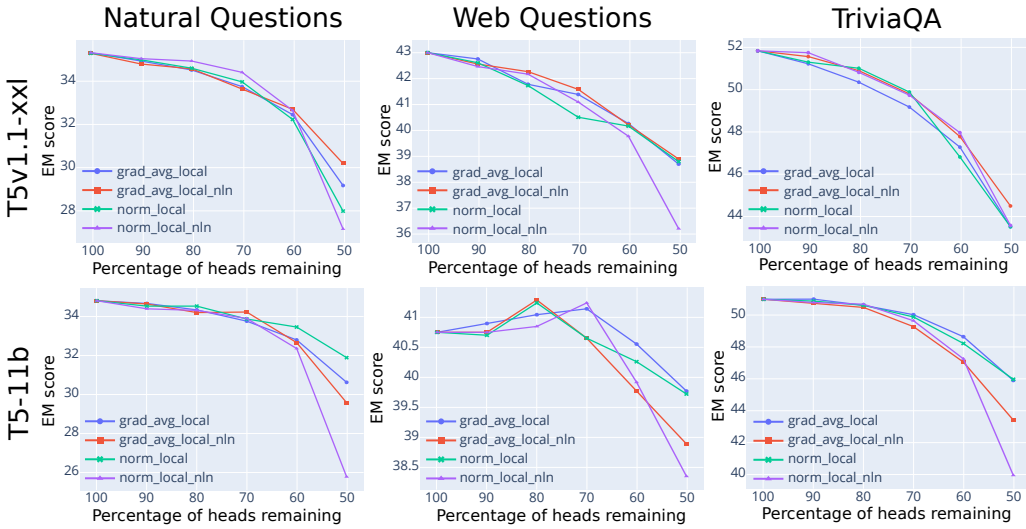

Figure 4.7: Least-important first pruning experiments on three datasets with T5-11b and T5v1.1-xxl models, using the best-performing settings and methods.

heads, layer normalization produces less uniform maps.

Figs. 4.6 and 4.7 show that for *grad_avg*, layer normalization becomes more important as the number of heads in a model increases. We find that non-normalized importance maps tend to assign more importance to later layers as models increase in size, and least important heads just become "heads of earlier layers." However, when normalized by layer, the least important heads from all layers are pruned. This is not a problem in small models because when the number of heads is small (e.g., 16), heads of medium importance are not condensed only in later layers. We focus on heads of medium importance here because we notice in Figure 4.7 that the sharpest drop between normalized and non-normalized experiments is when 50–60% of heads are remaining.

To understand the intuition of why performance does not decrease much at the beginning of the pruning, we point out that removing the least important heads in the *norm* method is akin to removing the least impactful $\mathbf{z}$ vectors, as vectors with a small norm value are closer to a zero vector. Intuitively, smaller vectors contribute less than longer ones in reaching the embedded location of an answer when combined. For the *grad_avg* method, the least changed heads mean

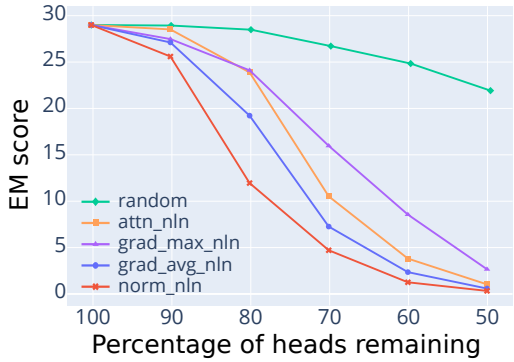

Figure 4.8: Pruning most important heads first with different methods using T5-large on the NQ dataset.

correcting the vectors in those heads was not necessary to find the correct answer. Therefore, they are probably not crucial for finding the answer.

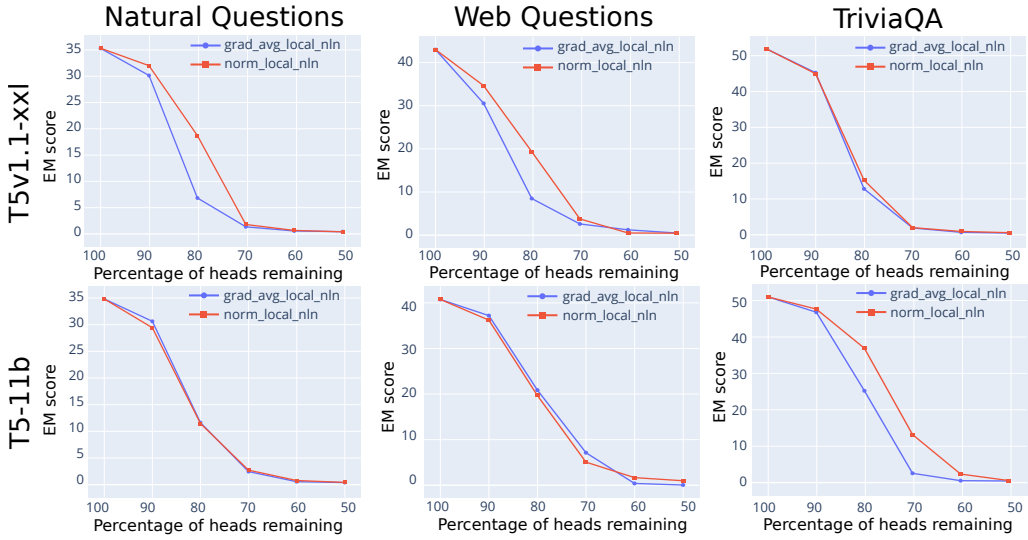

Figure 4.9: Most-important first pruning experiments on three datasets with T5-11b and T5v1.1-xxl models, using the best-performing settings and methods.

Most-important Heads First (MIF)

In this mode, we prune important heads first and expect to see a rapid drop in performance. We stop pruning when 50% of heads are pruned. We show a method

comparison in Figure 4.8 only with the "nln" setting, as we find that normalizing by layer loses valuable information about important heads. The *norm* method finds important heads better in T5-large; however, we find that, although this behavior is consistent between datasets, it differs with model changes. In T5-11b, *grad_avg* and *norm* have similar performance, while in T5v1.1-xxl, *grad_avg* drops faster than *norm*. We show the graphs in Figure 4.9. An exception to the pattern of T5-11b is the TriviaQA dataset, where the *norm* method performs worse. We think this is because the size of the evaluation set is over three times that of the other two datasets. This causes the *norm* scores of heads to be averaged more flatly over the 128 heads, with the data distribution being another possible factor.

As the main change in behavior (i.e., *grad_avg* and *norm* methods swapping places in which one drops the performance faster) is between T5-large and T5v1.1-xxl, we look at all the differences explained in Section 4.3.3. Then, list the main differences directly influencing the scoring methods between the models and discuss their likelihood of being the responsible change:

- Hidden model size (d_model): T5-large has d_model of $1,024$, while T5v1.1-xxl has $4,096$. While $\mathbf{Z}^h \in \mathbb{R}^{d\_kv}$ did not change between the models, the *grad_avg* was performed on $64 \times 4,096$ instead of $64 \times 1,024$, giving a better hint at what is important because of viewing more gradients, making this change likely responsible.

- The number of heads: T5-large has 16 heads, while T5v1.1-xxl has 64 heads. This increase in heads may not be the reason for the change because T5-11b has 128 heads, yet the two methods perform similarly.

- Feed-forward layers size (d_ff): For T5-large it is $4,096$, while for T5v1.1-xxl it is $10,240$. This change is not responsible for two reasons: T5-11b has a more significant FF layer size of $65,536$ with no drastic behavior change, and the gradients in the FF parameters are negligible, as shown in Figure 4.1.

Therefore, we conjecture that the reason for the behavior change is the d_model size. However, many experiments (changing each factor and observing the performance change) are required to prove all hypotheses. Because their computational

costs are unaffordable for a university research lab, this verification is left for future work.

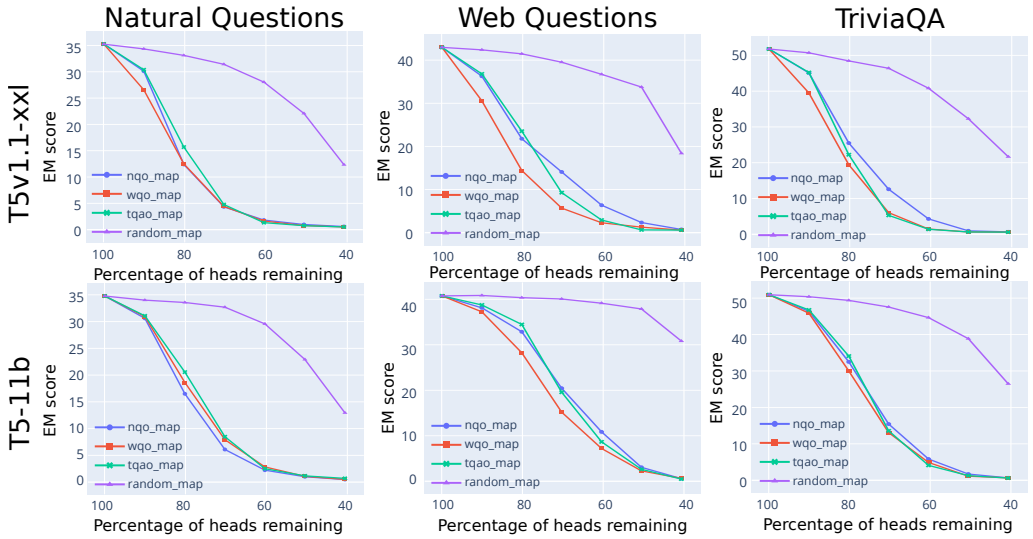

Figure 4.10: Most-important first pruning experiments on three datasets with T5-11b and T5v1.1-xxl models, using *grad_avg* with "nln" and the importance maps that were calculated on different datasets.

Head Similarity Between Datasets

Figure 4.5 showed that models have similar head importance maps across different datasets because of the underlying pre-training phase. We prove this similarity in this section.

We generate three importance maps for each model, one for each dataset (nqo_map, wqo_map, tqao_map) using the *grad_avg* method explained in Section 4.3.2. For example, nqo_map of T5-11b is a head importance map generated by evaluating the T5-11b on the NQ dataset, while the wqo_map is generated with the WQ dataset, and so on. We then apply important-first pruning on a model and a dataset using each of the generated maps (the one generated on the dataset itself and on other datasets as well) and compare the results against a baseline of random head importance, random_map. The results are shown in Figure 4.10. The important heads for one dataset are also important for the others because of

the shared knowledge in the pre-training phase.

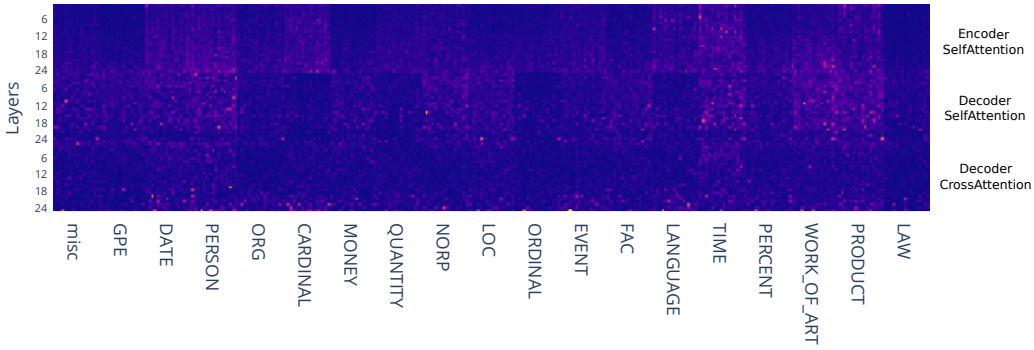

Figure 4.11: Distribution of knowledge inside T5-large attention heads depending on the named entity type of the target answer, calculated using the *grad_avg* method.

4.3.6 Distribution of Knowledge per Answer Type

In Section 4.3.5, we showed that the knowledge stored in T5 is not evenly distributed and that some attention heads store more knowledge than others. However, a question arises: are there heads that are specialized in certain types of questions? To investigate the distribution of different types of questions (who, where, when, etc.), we analyze the named entity type of the target answer (person, location, date, etc.).

We used SpaCy to do NER on all target answers, and we separated the *grad_avg* importance map with "nln" calculated in Section 4.3.4 by named entity type. The results are shown in Figure 4.11. Different named entities have different patterns of importance. To determine the similarity of importance maps between entity types, we sum the importance of each type per area (EncSA, DecSA, or DecXA).[5] Then, we calculate the Kullback-Leibler (KL) divergence between different entity types and show the results in Figure 4.12. We sort the entity types by similarity and notice the correlation with the KL-divergence. In general, similar entity types are stored in similar locations. For example, geopolit-

[5]We calculated divergence per head and layer, but the results were not meaningful, as one head or layer may not hold enough information about many types to deduce similarity.

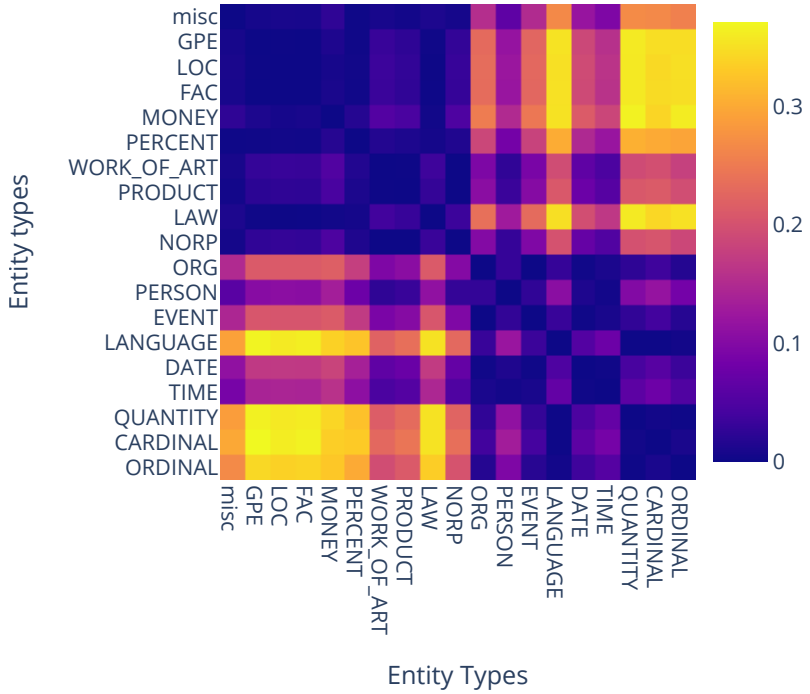

Figure 4.12: KL-divergence of importance maps per area between named entity types. Lower divergence represents higher similarity.

ical entity (GPE), location (LOC), and facilities (FAC) have similar distribution because they are essentially locations. CARDINAL, QUANTITY, and ORDINAL are similar because they are all related to numbers. DATE and TIME are both related to time.

We note that the results in Section 4.3 might be dependent on the three experimented datasets. In future work, we want to experiment with other datasets, including ones with different domains.

4.4 Related Work

4.4.1 Confirming the Existence of Knowledge

Several works investigating knowledge probing are surveyed by Liu et al. (2023): Petroni et al. (2019) have introduced the LAMA probe that tests the factual and common-sense knowledge in LMs. Using four knowledge sources, they convert

the fact triples or question-answer pairs into cloze templates, and they evaluate models on how highly they rank the ground truth token. They found that factual knowledge can be recovered "surprisingly well" from pre-trained LMs. Some work explored continuous template learning instead of discrete prompt template search. Also tested on the LAMA benchmark, Zhong et al. (2021) proposed a model OptiPrompt that optimizes in continuous embedding space to find prompts, rather than being limited to a space of discrete tokens.

Roberts et al. (2020) estimated how much knowledge is stored in LMs by evaluating T5 models on open-domain QA in a closed-book setting. Compared to other competing models, they show that LMs can achieve competitive results on open-domain QA benchmarks without access to context or external knowledge. Different from previous work, we investigate how Transformer-based T5 stores knowledge for closed-book QA.

4.4.2 Methods of Importance Scoring

Michel et al. (2019) found that a percentage of heads can be removed with no significant impact on the performance. Their criteria for assigning the importance of heads are explained in Section 4.3.2. They effectively give a higher importance score if the attention score is high, which, as presented in Section 4.3.5, is insufficient. Voita et al. (2019) also focused on the translation task and used several scoring methods: head "confidence" is the average of its maximum attention score over tokens in a sentence. As this method focuses on the "maximum" attention score, it will have similar problems as the *attn* method in Section 4.3.2. They also used a method called layer-wise relevance propagation (LRP), which has similar behavior to the "confidence" method. They also performed pruning experiments based on stochastic gates and relaxed L_0 penalty criteria. However, their pruning method requires fine-tuning to prune heads, and their results coincide with that of their LRP analysis. Hence, we exclude their methods from our comparison. In this work, we introduce other methods and compare them with the *attn* method.

While we only focus on attention head pruning, Prasanna et al. (2020) experimented with other techniques, such as magnitude pruning, where sparse weights are pruned according to their magnitude and whole layers pruning. However,

these techniques are not informative as they are either quite fine-grained or overly general.

4.5 Summary of This Chapter

This chapter provided insight into how Transformer-based T5 operates in a closed-book QA setting. We showed what components of the T5 are important in retrieving the factual knowledge within. We explored the importance of heads in retaining knowledge via pruning while using different settings and scoring methods. In the future, we will investigate other Transformer-based LMs, and how they work on NLP tasks with context, such as open-book QA.

Chapter 5

Improving Multi-Hop QA in Closed-Book Setting

Transformer-based language models (LMs) have been shown to perform question answering (QA) competitively even when removing context and using only questions as input (called closed-book QA). Previous work that studied closed-book has mainly used simple questions that require a single reasoning step (i.e., single-hop questions). In this study, we find that using multi-hop questions requiring multiple reasoning steps drastically drops the performance. We investigate how to close this gap using two methods: fine-tuning with explicit question decomposition using three decomposition systems, or few-shot learning with chain-of-thoughts (CoT) for implicit question decomposition. We experiment on three multi-hop datasets, considering different multi-hop questions types (i.e., compositional, comparison, etc.). We demonstrate when the methods fail and identify future directions that are most promising to closing the gap between single-hop and multi-hop closed-book QA. We will make the code publicly available.

Closed-book QA is a setting where only questions, without accessing context or external knowledge, are used to obtain answers (Roberts et al., 2020). Models must answer using the knowledge they stored in their parameters during pre-training. Recent work shows that LMs in a closed-book setting achieve competitive performance on single-hop datasets (questions requiring a single reasoning step) compared to open-book (i.e., using context) (Roberts et al., 2020).

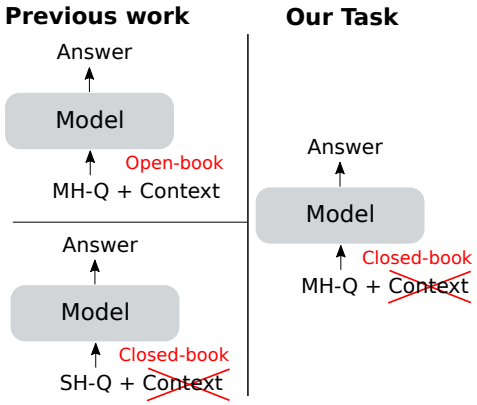

Figure 5.1: Overview of our task. SH-Q is a single-hop question, while MH-Q is a multi-hop question.

For multi-hop questions (requiring multiple reasoning steps), previous work experiments in an open-book setting (Fang et al., 2020, Alkhaldi et al., 2021). However, it is unclear if multi-hop questions have a similar trend to single-hop questions in a closed-book setting.

This study aims to investigate the gap between single-hop and multi-hop questions in a closed-book setting using three representative multi-hop datasets. We compare this task with previous work in Figure 5.1. As single-hop performance is found to be considerably higher, we attempt to improve the accuracy for multi-hop questions by decomposing into single-hop sub-questions (SQ) explicitly, or implicitly via generating CoT before answering; this should facilitate finding which reasoning steps to perform. To the best of our knowledge, this is the first study that attempts to improve the accuracy of multi-hop questions in a closed-book QA setting. We investigate closing the gap between single-hop and multi-hop questions in a closed-book setting by explicitly including the reasoning steps with the question in the shape of single-hop decompositions or CoT, using a proposed fine-tuning method or a few-shot learning method, and compare results on three datasets.

5.1 Related Work

Jiang et al. (2022) also investigate multi-hop reasoning via decomposition; however, different from our work, they use golden decompositions for training and dev splits, and focus on T5. We experiment with three decomposers in addition to gold SQs and their answers. We also experiment on GPT-3 in addition to T5. Our T5 experiments are not directly comparable to theirs because, in addition to using different datasets, the setting difference; they fine-tune models with single-hop and multi-hop questions, then answer SQs sequentially and compare with answering multi-hop questions. Also, they fine-tune on concatenated SQs to show that they are a good approximation to fine-tuning on multi-hop questions. The improvement they attempt is on a *single-hop* baseline, while we aim to improve the *multi-hop* baseline.

Previous work on CoT (Wei et al., 2022; Kojima et al., 2022) experiment on datasets that mainly do not require external factual knowledge to achieve answers (e.g., math or multiple-choice datasets).

5.2 Task

We do fine-tuning and inference on multi-hop questions without context (i.e., closed-book). When a multi-hop question is given as input to an LM, it generates an answer text. We use sequence-to-sequence (Seq2Seq) LMs whose inputs and outputs are text, as opposed to LMs without a decoder like BERT (Devlin et al., 2019). Seq2Seq models are necessary for a closed-book QA setting because BERT-like models assume the availability of context from which to select text spans as output, while Seq2Seq models can generate text. As no context is provided, Seq2Seq LMs generate answers by looking only at questions.

5.3 Decomposers

DecompRC Introduced by Min et al. (2019), this method relies on selecting text spans from the question to form SQs. They train span prediction models using 400 annotations and show that it achieves reasonable performance. Different

question types have different models. They use trained models for CL and I types, while heuristic rules for the C type.

MuSiQue-based Decomposer We train a T5-large[1] (Raffel et al., 2020) to learn the gold decompositions provided by MuSiQue with 14,376 and 1,252 train and dev questions, respectively. The input is the full question, while the target is the two SQs. Example input: `What year saw the creation of the region where the county of Hertfordshire is located?` Target: `[sq1] In which state is Hertfordshire located? [sq2] When was #1 birthed?` The accuracy on the held-out split is 79.6 F1. This decomposer only supports CL type as MuSiQue only contains CL questions. We also compare against using gold MuSiQue decompositions in fine-tuning and inference and using intermediate gold answers to replace the answer placeholder.

ModularQA Introduced by Khot et al. (2021), it is a next-question generation (NQG) system that sequentially generates SQs given the full question and previous SQs with their predicted answers. They first train a model to generate questions given context, the answer, and hints that are keywords from the question. Then using this model, they generate decompositions for multi-hop questions that could lead to likely intermediate answers. They use these SQ sequences to train the NQG. At inference time, the NQG first generates SQ1, then a single-hop-trained sub-model answers it. Then, the full question, SQ1, and its answer are provided to NQG to produce SQ2. In CL question types, SQ2 includes the answer of SQ1, so we find the answer and replace it with #1 placeholder. This decomposer is not limited to any question type.

5.4 Method

An overview of the baseline, our proposed fine-tuning method, and a few-shot CoT method are in Figure 5.2.

[1]We also tried T5-11b but achieved similar accuracy.

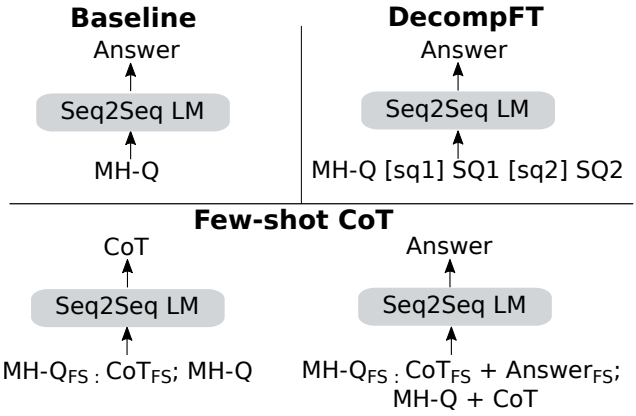

Figure 5.2: Methods overview. SQ1 and SQ2 are the decomposed SQs, `[sq1]` and `[sq2]` are special tokens.

5.4.1 Baseline

We use multi-hop questions as input to the LM, and answers as targets for fine-tuning and evaluating $\text{Baseline}_{\text{FT}}$, and for prefixing (question, answer) pairs as few-shot examples for evaluating another $\text{Baseline}_{\text{FS}}$.[2] We do not use any context or external knowledge. This is the basic evaluation setting for closed-book QA.

5.4.2 Decomposition Fine-Tuning (DecompFT)

In the pre-processing step, we decompose the multi-hop question into two single-hop SQs: SQ1 and SQ2, using one of the decomposers explained in Section 5.3. If SQ2 depends on the answer of SQ1, the answer is replaced by the placeholder #1. Then both SQs are concatenated with the full question to form the input for the LM. An example is in Table 5.1.

5.4.3 Few-shot CoT In-Context Learning (CoT)

We prefix multi-hop questions with few-shot examples of (MH-Q_{FS} : CoT_{FS}) pairs, similar to (Wei et al., 2022), in the following format: {`Q: MH-`Q_{FS}`\nA:CoT`$_{\text{FS}}$`...`}. However, as our target is a free-form text, models had difficulty constantly gen-

[2]We tried zero-shot mode but without few-shot examples, the LM gave a full sentence as the answer, mismatching the target. Therefore, we exclude zero-shot experiments.

DecompFT
Input: When was the university Elizabeth Harwood attended formed? [sq1] What university did Elizabeth Harwood attend? [sq2] When was #1 formed? **Output**: 1972
FS CoT - step 1
Input: Q: When was the institute that owned The Collegian founded? Let's think step by step. A: The Collegian was owned by Houston Baptist University. Houston Baptist University was founded in 1960. ... Q: Who is the spouse of the Rabbit Hole's producer? Let's think step by step. A: **Output**: The producer of Rabbit Hole is Nicole Kidman. Nicole Kidman's spouse is Keith Urban.

Table 5.1: Example of Input/Output for DecompFT and few-shot (FS) CoT methods. A full example for CoT is in Table 5.2.

erating parsable short answers.[3] Therefore, we follow Kojima et al. (2022) in applying a two-step prompting. One for generating CoT with the multi-hop question prefixed with few-shot examples, as explained above, and another for generating the answer with each few-shot example appended its answer, and the CoT generated in the first step appended to the multi-hop question, as shown in Figure 5.2. In the first prompting step, the few-shot examples are of the format: `Q: <Multi-hop example question1> Let's think step by step.\nA: <CoT>`, before appended with our question as `Q: <MH-Q>\nA:`. Then for the second step, we add after every few-shot example's Cot the phrase: `Therefore, the answer is` followed with the target for the few-shot examples. We show an example of the few-shot prompt in Table 5.2. We used 4 few-shot examples, with temperature 0 for decoding. [4]

[3] Previous work experimented on datasets whose answers were numerical, yes/no, or multiple choice.

[4] We used OpenAI API with *Completion* mode.

First prompting step	Second prompting step
Q: When was the institute that owned The Collegian founded? Let's think step by step. A: The Collegian was owned by Houston Baptist University. Houston Baptist University was founded in 1960.	Q: When was the institute that owned The Collegian founded? Let's think step by step. A: The Collegian was owned by Houston Baptist University. Houston Baptist University was founded in 1960. Therefore, the answer is 1960.
Q: Jan Šindel's was born in what country? Let's think step by step. A: Jan Šindel's birthplace is Hradec Králové. Hradec Králové is in Czech Republic.	Q: Jan Šindel's was born in what country? Let's think step by step. A: Jan Šindel's birthplace is Hradec Králové. Hradec Králové is in Czech Republic. Therefore, the answer is Czech Republic.
Q: What city is the person who broadened the doctrine of philosophy of language from? Let's think step by step. A: The person who broadened the doctrine of philosophy of language is Søren Kierkegaard. Søren Kierkegaard is from the city of Copenhagen.	Q: What city is the person who broadened the doctrine of philosophy of language from? Let's think step by step. A: The person who broadened the doctrine of philosophy of language is Søren Kierkegaard. Søren Kierkegaard is from the city of Copenhagen. Therefore, the answer is Copenhagen.

Q: Who was thee first president of the association that wrote the code of ethics for psychology? Let's think step by step. A: The American Psychological Association wrote the code of ethics for psychology. The first president of The American Psychological Association is G. Stanley Hall.	Q: Who was thee first president of the association that wrote the code of ethics for psychology? Let's think step by step. A: The American Psychological Association wrote the code of ethics for psychology. The first president of The American Psychological Association is G. Stanley Hall. Therefore, the answer is G. Stanley Hall.
Q: Who is the spouse of the Rabbit Hole's producer? Let's think step by step. A: **The producer of Rabbit Hole is Nicole Kidman. Nicole Kidman's spouse is Keith Urban.**	Q: Who is the spouse of the Rabbit Hole's producer? Let's think step by step. A: The producer of Rabbit Hole is Nicole Kidman. Nicole Kidman's spouse is Keith Urban. Therefore, the answer is **Keith Urban**

Table 5.2: Example of the two-step prompts for few-shot CoT. The text in **bold** is the output of GPT-3 Davinci model.

5.5 Experiments

5.5.1 Datasets

We use three multi-hop datasets: HotpotQA (Yang et al., 2018), constructed by crowdsourcing 2-hop questions based on two connected Wikipedia articles; 2WikiMultiHopQA (Ho et al., 2020) (2Wiki), constructed from structured and unstructured data of Wikipedia using predefined templates; and MuSiQue (Trivedi et al., 2022), 2-4 hop questions made from seed questions from 5 existing single-hop datasets. We only use 2-hop questions as other datasets are only 2-hop. MuSiQue

Question Type	HotpotQA		2Wiki		MuSiQue	
	train	dev	train	dev	train	dev
Compositional / Bridging	(*) ∼49,008	(*) 2,509	76,481	5,236	14,376	1,252
Comparison	17,456	1,487	51,963	3,040	–	–
Bridge-Comparison	–	–	34,631	2,751	–	–
Intersection	(*) ∼23,983	(*) 1,239	–	–	–	–
Inference	–	–	4,379	1,549	–	–
Total	90,447	(*) 5,235	167,454	12,576	14,376	1,252

Table 5.3: Data statistics for all question types and datasets. (*) As HotpotQA only has "Bridge" and "Comparison" labels, we use the approximate percentage of labels "Bridging" and "Intersect" (both under "Bridge" label in HotpotQA) provided by Min et al. (2019) to calculate the number of training questions. For the "Bridge" dev split, we use the labels classified by Min et al. (2019), while excluding questions classified as other types.

also includes gold decompositions and intermediate answers for SQ1. We show statistics in Table 5.3.

5.5.2 Question Types

Datasets have different multi-hop question types, with some common types across datasets. We list the types.

Compositional/Bridging (CL) It is composed of two questions connected by a bridge entity. Finding the entity is required to solve the question. Example: *Where was the director of film Ronnie Rocket born?* MuSiQue consists only of this type.

Comparison (C) Requires comparing two entities, where the two entities are mentioned explicitly in the question. Example: *Who died first, Fleetwood Sheppard or George William Whitaker?*

Bridge-Comparison (BC) Similar to the previous type, but requires finding the entities first before comparing them. Example: *Which film has the director born first, Once A Gentleman or The Girl In White?* This type only exists in the 2Wiki dataset. We consider this to be in the same category as "Comparison" because of similarity.

Intersection (I) Requires finding an entity that satisfies different conditions. Example: *Which college football team for the University of Oregon did Enoka Lucas play for?* This type only exists in HotpotQA. Min et al. (2019) identified this type and only classified the dev split accordingly. As both "Bridging" and "Intersection" types are labeled as "Bridge" in HotpotQA, we fine-tune on them together but evaluate CL and I separately.

Inference (Inf) Requires inferring the bridge entity using common-sense and logical rules. It is different than the CL type in that it is not explicit to find the bridge entity. Example: *Who is the paternal grandmother of King Kang Of Zhou?* This type only exists in the 2Wiki dataset.

PERSON	DATE	ORG	misc	GPE	CARDINAL
475	139	130	114	113	106
LOC	NORP	WORK_OF_ART	PERCENT	EVENT	ORDINAL
40	33	31	17	15	10
FAC	QUANTITY	LAW	TIME	MONEY	–
8	7	6	4	2	–

Table 5.4: Statistics about named entity types of answers for the dev split of MuSiQue dataset.

5.5.3 Selecting Few-shot Examples Based on Answer Types

Named entity types of answers correspond to the type of the WH question. For example, answers of type (PERSON, LOCATION, DATE, etc.) correspond to (who, where, when, etc.) questions. For the few-shot examples explained in Section 5.4.3, we select them to represent different types of answers, and we use that few-shot set with all questions. An alternative way is to select the few-shot examples based on the question. Questions about people would have few-shot examples with answers of type *person*, while a question about locations would have examples with answers of type *GPE*, etc.

To test if such a setting improves the performance, we apply this technique by selecting several few-shot example groups based on the *gold* answer entity type. We did not use a classifier on the question because we want to check the upper bound. We use the SpaCy framework to do named entity recognition on the answers of the dev split of the MuSiQue dataset, we report the statistics of entity types in Table 5.4.

We collect few-shot example groups for the following types: PERSON, DATE, GPE, ORG, CARDINAL. If the answer entity type is other than those types, we use the mixed few-shot example group used in the other experiments.

5.5.4 Experimental Settings

The used models are a pre-trained T5-large (T5-L) with 770M parameters, and two GPT-3 versions (Brown et al., 2020a); Curie and Davinci with 13B and 175B

parameters, respectively. We only fine-tune T5-L and apply few-shot prompting to GPT-3. We evaluate both models on three datasets, and we use two accuracy metrics: EM = exact match, and F1 = partial match.

We manually search for hyper-parameters, and after some trials, we run all experiments using the best-found ones. We used a learning rate of 5e-5 and a batch size of 64, except for MuSiQue; because of the small size, we used a batch size of 8 and a learning rate of 1e-5. We found that a high learning rate like 1e-3 harms performance. We used Adafactor optimizer with no learning rate decay or warm-up, and a max gradient norm of 1.0. We used early stopping, where we stop training after 15 epochs of no improvement in the validation score. We report scores of single runs. We did not fine-tune T5-11b for all experiments due to computational limitations, instead, we report some T5-11b results in Table 5.7. Fine-tuning T5-L time depends on the dataset and the GPU type, but an experiment, on average takes approximately 10 hours. We mostly used NVIDIA GeForce RTX 3090 and Tesla A100 GPUs.

5.5.5 Results

Roberts et al. (2020) evaluated three single-hop datasets in a closed-book setting: NaturalQuestions (Kwiatkowski et al., 2019), WebQuestions (Berant et al., 2013a), and TriviaQA (Joshi et al., 2017). T5-L achieved 28.5, 30.6, and 28.7 EM, respectively. We consider these as a reference for single-hop closed-book performance.

From Table 5.5, we see an expected large gap in performance between multi-hop and single-hop baselines in types CL and Inf for all models, and in type I for T5-L and Curie models, demonstrating the difficulty of multi-hop questions. An exception is the C type; the answer is present in the question itself, therefore the LM can generate it with ease.

Summary of Findings We find that even though GPT-3 is more than two orders of magnitude larger than T5-L, fine-tuning T5-L is superior under two conditions: enough training data is provided, and the questions are truly multi-hop and not guessable from one-hop like type I questions. In addition, we find

Data Split	Model	$\text{Baseline}_{\text{FT/FS}}$	DFT or CoT
HotpotQA (CL)	T5-L	8.45 / 15.94	8.65 / 16.51
	Curie	7.25 / 14.49	4.94 / 10.50
	Dvnc	17.90 / 27.32	**22.52 / 34.28**
2Wiki (CL)	T5-L	**16.31 / 18.62**	**16.31** / 18.56
	Curie	2.90 / 5.48	1.99 / 3.97
	Dvnc	6.46 / 10.65	9.19 / 15.82
MuSiQue (CL)	T5-L	2.40 / 8.48	3.22 / 9.02
	Curie	2.80 / 9.56	2.24 / 6.84
	Dvnc	6.56 / 15.31	**14.48 / 25.05**
HotpotQA (C)	T5-L	53.87 / 60.38	52.80 / 58.77
	Curie	36.32 / 43.57	35.64 / 40.57
	Dvnc	51.11 / 58.73	**60.93 / 68.94**
2Wiki (C+BC)	T5-L	50.56 / 52.05	50.26 / 51.68
	Curie	43.31 / 44.31	42.86 / 44.11
	Dvnc	44.78 / 45.22	**58.95 / 59.31**
HotpotQA (I)	T5-L	11.22 / 19.63	10.82 / 19.54
	Curie	10.57 / 18.97	7.022 / 14.44
	Dvnc	25.02 / 37.35	**25.75 / 38.82**
2Wiki (Inf)	T5-L	**10.85 / 30.99**	10.40 / 29.38
	Curie	1.81 / 21.50	1.55 / 21.08
	Dvnc	5.62 / 27.47	8.65 / 30.45

Table 5.5: Comparing the baselines against DecompFT/CoT methods. For T5-L rows, $\text{Baseline}_{\text{FT}}$ and DecompFT (DFT) are used, while for Curie/Davinci rows, $\text{Baseline}_{\text{FS}}$ and few-shot CoT are used. For DecompFT, we used the highest-performing decomposition method. Full DecompFT results in Table 5.6. All results are shown in EM / F1 format. We group the question type splits of the datasets together.

that predicting a correct bridging entity/answer to SQ1 (we call it SA1) is more important than correct SQs. We explain more in detail.

Analysis of Baselines Davinci's high performance for type I is because of the nature of questions that allow the answers to be guessed without other reasoning hops (like single-hop questions). For example: *Based on a True Story is an album by which country music star, with the single My Eyes?*. The T5-L performance on MuSiQue is especially low due to small training data. Another factor is that the dataset was designed to be harder by reducing shortcuts and artifacts that occur in other models (Trivedi et al., 2022). The 2Wiki Inf split is smaller, however, upon investigation, most question formats are: *Who is the grandparent of X?* and a large portion of the answers overlap greatly with the question. This explains the high F1 score compared to other splits. The high T5-L performance for 2Wiki CL split is because of the sufficiently large training data. The high performance of GPT-3 on the HotpotQA CL split is likely due to wrongly included type I questions because we used the type prediction by Min et al. (2019), as mentioned in Section 5.5.2. For types C and Inf, the T5-L baseline outperforms GPT-3 because of fine-tuning.

Analysis of DecompFT We discuss adding SQs to help the model explicitly do multi-hop reasoning. As a quantitative quality analysis of the question decompositions, we compare DecompFT using different decomposers with $\text{Baseline}_{\text{FT}}$ in Table 5.6, and we see no improvement, and sometimes degradation due to decomposition failures; DecompRC failed to decompose C type for 2Wiki 28.5% of the time. This suggests that the quality of predicted SQs is not sufficient.

However, when we use gold decompositions, we see a boost, albeit small, in Table 5.7. To test DecompFT with predicted SA1, we train a single-hop T5-L and T5-11b using gold SQ1 and SQ2 scoring 6.15 / 14.07 and 16.65 / 25.51, respectively. Table 5.7 shows that plugging low-quality predicted SA1 to the gold decompositions hurts the performance, while T5-11b's better quality SA1 improves it. Gold SA1 expectedly provides remarkable improvement where the scores outperform the single-hop accuracy, confirming that the DecompFT method

Dataset split	Baseline	MuSiQue-based	DecompRC	ModularQA
HotpotQA (CL)	8.45 / 15.94	8.29 / 15.71	**8.65** / **16.51**	7.94 / 16.03
2Wiki (CL)	**16.31** / **18.62**	**16.31** / 18.56	16.25 / 18.40	15.10 / 17.24
MuSiQue (CL)	2.40 / 8.48	2.88 / **9.35**	2.80 / 8.56	**3.22** / 9.02
HotpotQA (C)	**53.87** / **60.38**	–	52.80 / 58.77	52.69 / 59.36
2Wiki (C+BC)	**50.56** / **52.05**	–	47.85 / 48.89	50.26 / 51.68
HotpotQA (I)	**11.22** / **19.63**	–	10.49 / 18.96	10.82 / 19.54
2Wiki (Inf)	**10.85** / **30.99**	–	–	10.40 / 29.38

Table 5.6: Results of comparing the baseline against DecompFT using different decomposers on three datasets. All results are shown in EM / F1 format.

Setting	T5-L	T5-11b
No decomps	2.40 / 8.48	4.31 / 12.20
+ gold decomps	3.35 / 9.64	5.03 / 12.66
+ pred SA1	2.56 / 8.33	5.91 / 14.34
+ gold SA1	**7.19 / 16.39**	**18.53 / 29.80**

Table 5.7: Comparison with using gold decompositions on MuSiQue (CL), and between using predicted and gold SA1, all in both fine-tuning and inference.

is viable, and the importance of SA1.

A more fine-grained analysis is possible on the MuSiQue dataset because we have golden decompositions and gold SA1. For our trained decomposer using the MuSiQue-provided decompositions (Section 5.3), we evaluate the generated SQ1 for the dev split of MuSiQue on our single-hop fine-tuned model, we get the following scores: 4.71/10.31. The scores for the gold SQ1 on the same split and single-hop model are 7.27/12.87. As a qualitative analysis, we show a sample of the decompositions in Table 5.8 on the MuSiQue split, as it has golden decompositions which can be used as reference.

The boost of using gold decompositions without SA1 in Table 5.7 could be higher as about half of the gold SQs are not in natural language, but in a shortened format, which might hurt the performance of our trained decomposer and the oracle results.

Gold Decomp	MuSiQue-based Decomp	DecompRC Decomp	ModularQA Decomp
Q: Who is the spouse of the Green performer?			
SQ1: Green >> performer SQ2: #1 >> spouse	SQ1: Green >> performer SQ2: #1 >> spouse	SQ1: which spouse of the Green performer? SQ2: Who is [ANSWER]	SQ1: Who was the Green performer? SQ2: Who was #1' spouse?
Q: Which company owns the manufacturer of Learjet 60?			
SQ1: Learjet 60 >> manufacturer SQ2: #1 >> owned by	SQ1: Learjet 60 >> manufacturer SQ2: #1 >> owned by	SQ1: which manufacturer of Learjet SQ2: Which company owns [ANSWER] 60?	SQ1: Who was the manufacturer of the Learjet 60? SQ2: What company owns #1?
Q: The Unwinding author volunteered for which organisation?			
SQ1: The Unwinding >> author SQ2: #1 >> member of	SQ1: Unwinding >> author SQ2: #1 volunteered for what organisation?	SQ1: The SQ2: [ANSWER] Unwinding author volunteered for which organisation?	SQ1: What is the author of Unwinding? SQ2: What has #1 volunteered for?

Table 5.8: Example of the quality of question decompositions of different decomposers.

Impact of CoT's Quality CoT, by default, includes the equivalent of SA1 (i.e. the bridging entity). Curie's generated CoT degrades its performance. Upon examining the quality of the generated CoT, we found that the problem mainly is not with the logic, as the case with the previously experimented decomposers, but rather with the bridging entity. Predicting the wrong entity harshly harms the performance, as seen in Table 5.5. However, while the phrasing of Davinci's CoT is similar to that of Curie's, the bridging entities are different. Predicting the correct entity essentially transforms the original question into a single-hop one, increasing the performance. Nonetheless, Curie occasionally makes logical mistakes in comparison steps and fails to decompose the question.

We show a comparison of the generated CoT by Curie and Davinci models in Table 5.9. We see that in 2W (CL), MSQ (CL), HP (I), and 2W (Inf), Curie's reasoning steps begin correctly, but it fails to produce the bridging entity or last answer correctly; which is a fault of the factual knowledge capability of Curie. Curie also struggles with comparison logic; in HP (C), it predicts the information about the entities correctly but fails to know that `American == American`. In 2W (C+CB), Curie's logic starts good but it predicts wrong entities, and at the end, it compares when movies were *made* instead of the birth date of the directors.

Data Split		CoT output
HP (CL)	Curie	Q: The director of the romantic comedy "Big Stone Gap" is based in what New York city? (Gold answer: Greenwich Village, New York City) A: The director of the romantic comedy "Big Stone Gap" is based in New York City. New York City is in the United States of America.
	Davinci	A: **Adriana Trigiani** is the director of the romantic comedy "Big Stone Gap". **Adriana Trigiani** is based in New York City.
		Q: What nationality is the director of film World And Time Enough? (Gold answer: United States)

2W (CL)	Curie	A: The director of film World And Time Enough is a British national.
	Davinci	A: The director of film World And Time Enough is **John Frankenheimer**. **John Frankenheimer** is American.
MSQ (CL)		Q: The Unwinding author volunteered for which organisation? (Gold answer: Peace Corps)
	Curie	A: The Unwinding author, **George Packer**, volunteered for the New York Times. **George Packer** volunteered for the New York Times in the early 2000s.
	Davinci	A: The Unwinding author is **George Packer**. **George Packer** volunteered for the Peace Corps.
HP (C)		Q: Were Scott Derrickson and Ed Wood of the same nationality? (Gold answer: yes)
	Curie	A: Scott Derrickson was of American nationality. Ed Wood was of American nationality. Scott Derrickson and Ed Wood were not of the same nationality.
	Davinci	A: Scott Derrickson is an American film director, screenwriter and producer. Ed Wood was an American filmmaker, actor, writer and editor. Scott Derrickson and Ed Wood were both American.
2W (C+CB)		Q: Which film has the director who was born later, El Extraño Viaje or Love In Pawn? (Gold answer: yes)
	Curie	A: The director of the film El Extraño Viaje is **Alejandro González Iñárritu**. He was born in 1969. The director of the film Love In Pawn is **Abbas Kiarostami**. He was born in 1941. El Extraño Viaje was made later than Love In Pawn.

	Davinci	A: El Extraño Viaje was directed by **Fernando Fernán Gómez**. He was born on December 5, 1921. Love In Pawn was directed by **George Cukor**. He was born on July 7, 1899. **Fernando Fernán Gómez** was born after **George Cukor**.
HP (I)		Q: What American professional Hawaiian surfer born 18 October 1992 won the Rip Curl Pro Portugal? (Gold answer: John John Florence)
	Curie	A: Kelly Slater won the Rip Curl Pro Portugal in 2016. Kelly Slater is an American professional Hawaiian surfer.
	Davinci	A: John John Florence is an American professional Hawaiian surfer born 18 October 1992. He won the Rip Curl Pro Portugal.
2W (Inf)		Q: Who is Catherine Of Pomerania, Countess Palatine Of Neumarkt's father-in-law? (Gold answer: Rupert)
	Curie	A: Catherine Of Pomerania, Countess Palatine Of Neumarkt was the wife of **John I, Count Palatine Of Neumarkt**. **John I, Count Palatine Of Neumarkt** was the son of Otto I, Count Palatine Of Neumarkt.
	Davinci	A: Catherine Of Pomerania, Countess Palatine Of Neumarkt was the wife of **John I, Count Palatine Of Neumarkt**. **John I, Count Palatine Of Neumarkt**'s father was Rupert I, Elector Palatine.

Table 5.9: Comparison of the generated CoT's quality between Curie and Davinci. Bridging entities are highlighted in **bold**.

Analysis of NER-based Few-shot Examples We find that using different few-shot examples for different questions, as explained in Section 5.5.3, improves

the Curie with CoT performance on the MuSiQue dev split to 2.56 / 9.25, and the Davinci with CoT to 14.96 / 25.69. The reason for the great improvement in the F1 for Curie but only slight improvement for Davinci is that specialized few-shot examples were helpful to Curie as it already had bad CoT quality so those specialized examples helped to get better CoT. Davinci is already good enough to generate good CoT, so those specialized examples did not help much.

5.6 Summary of This Chapter

We revealed a large gap between single-hop and multi-hop questions in the closed-book QA setting. We proposed two methods, fine-tuning question decompositions, or few-shot generation of CoT to facilitate teaching LMs to "hop," and find answers. We demonstrated that decomposing multi-hop questions without or wrongly predicting the bridging entity does not improve performance, while correctly predicting it boosts the performance. We also found that when enough data is provided, fine-tuning a small model like T5-L is enough to outperform few-shot GPT-3 on real multi-hop questions. In the future, we will investigate more effective ways of selecting few-shot examples for the few-shot CoT method.

Chapter 6

Conclusion

6.1 Summary

In this thesis, we addressed three QA topics: Improving multi-hop QA in an open-book setting, investigating model workings in a closed-book setting, and improving multi-hop QA in a closed-book setting.

In Chapter 3, we proposed ideas to improve answering multi-hop questions in an open-book setting. The goal of the task is to identify both the answer and supporting facts from multiple paragraphs that are necessary to answer the multi-hop question. We first identified the supporting facts and tagged them, keeping the remaining context accessible for the model. Then we tagged the keywords linking different paragraphs together. With joint training of the module for finding supporting facts and the answering module, we were able to raise the accuracy by allowing the model to flexibly focus on the detected supporting facts, withstanding possible erroneous fact predictions, and additionally focus on possible bridging entities to find the answer.

In Chapter 4, we showed experiments on the inner workings of the Transformer-based T5 model in a closed-book QA setting. After corrupting components with noise and observing the outcome, we found that the query component in the attention mechanism is the most responsible for finding the answer. Moreover, we pruned parts of the encoder and decoder in turns and monitored the differences. We concluded that the encoder was responsible for finding the broad entity em-

bedding, while the decoder found the exact entity. Furthermore, using different head-scoring techniques, we demonstrated that certain areas in the model retain more information about factual knowledge than others. Also, we showed that related entity types are generally clustered in similar areas in the model.

In Chapter 5, we investigated the gap between single-hop and multi-hop questions in a closed-book setting and improved answering multi-hop questions through explicit and implicit question decomposition. The goal of the task is to improve the multi-hop QA baseline in a closed-book setting. We learned from Chapter 3 that showing the reasoning steps explicitly is beneficial to the answering model. We applied this finding to a closed-book setting and improved the performance. We also identified several possible points for further improving the multi-hop baseline.

6.2 Future Work

6.2.1 Flexibly Focusing with Virtual GNNs

In this thesis, we used Longformer as the base of our reader and supporting facts finder modules for the open-book experiments. Recent research using GNNs like HGN (Fang et al., 2020) has proved to be effective at modeling relations between text segments; however, using the simpler Longformer models lacks using these features. An interesting way to include these features in our simpler architecture is to use global attention on special tokens representing different textual granularity to act as virtual GNNs, mimicking the actual GNNs used by Fang et al. (2020). This would allow for easier and more efficient focusing flexibly on the supporting facts using our proposal, as in addition to computational overhead, there would be architecture changes needed to distinguish the predicted supporting facts to focus on them, as opposed to our method, which tags them with special tokens.

6.2.2 Investigating Reasoning Paths in Closed-book QA

In this thesis, we visualized the Transformer heads when a single-hop question was provided to the Transformer-based T5 model in a closed-book setting. We established that the information retaining factual knowledge is concentrated in

certain areas in the model and found which parts are most responsible. For multi-hop questions, an appealing research direction is to investigate the visualization and responsible parts for the reasoning path, which consists of a decomposed multi-hop question into multiple single-hop questions. Also, identifying if the embeddings of the intermediate entity are triggered in the process of answering the multi-hop question would be an insightful study.

6.2.3 Investigating Important Attention Components in Other Tasks and Their Applications

After demonstrating the importance of the Q component of attention in the context of closed-book QA, a natural question about the applicability of this result to other tasks arises. For closed-book QA, the role of Q is to guide the model to correctly combine the V values of the input question tokens to point at the answer's embedding space. It is difficult to imagine the role of Q when the task is open-book QA or a completely different task like document summarization or sentiment analysis. Pinpointing important components allow for several benefits, including building more optimized architectures. For example, one future direction for closed-book QA could be training a model with larger Q and K values, which might enhance the performance. Alternatively, Keeping Q and K values but using a smaller V might decrease the parameter's count without performance loss. Other tasks might reveal different component behaviors, unlocking new recommendations.

6.2.4 Improving CoT via Question Decomposition and Vice Versa

With the importance of providing the reasoning steps of the answer in the form of decompositions or CoT, an intriguing future investigation is to evaluate if adding the automatically decomposed questions to the input for GPT-3 would improve CoT's quality and the final answer and if including CoT in the fine-tuning of T5-L (including training a CoT generator in a supervised way) would yield improvements.

6.2.5 Combining a Knowledge Model with a Reasoning Model

We observed in our work that predicting the intermediate entity correctly gives a large boost to answering the multi-hop question in a closed-book setting. Compared to smaller models like T5-L, models like GPT-3 possess a larger amount of factual knowledge but are harder to fine-tune. We found that, given enough data, fine-tuning the much smaller T5-L outperforms few-shot GPT-3. Considering these findings, an interesting future direction is to let an expert in factual knowledge (like GPT-3) provide the intermediate answer for decomposed questions while a fine-tuned reasoning expert takes the decompositions and the intermediate answer to find the final answer. The high accuracy of the intermediate answers would provide the large boost observed in our work.

6.2.6 Investigating the inners of generative large language models (LLMs)

Despite the recent popularity of generative LLMs, they are still considered blackboxes, as their inner workings and decision-making process are poorly understood. Also, although these models are capable of many tasks, the way they decide on the task based on instructions is still unclear. Shedding the light on the inner workings of these powerful models transforms them from blackboxes into interpretable and trustable tools.

An important aspect of the investigation involves exploring the instruction-related embedding spaces and parameters within LLMs, which has the potential to unlock valuable insights for facilitating transfer learning between tasks.

Another essential aspect to investigate is the internal decision-making process of LLMs. This process can be traced by identifying intermediate keywords of sample examples, then attempting to find where the embeddings of these keywords are triggered inside the model. By gaining a deeper understanding of how these models arrive at their predictions, we can gain valuable hints and insights into when to trust their outputs. This knowledge is invaluable, particularly in real-world applications where the reliability and interpretability of model predictions are crucial.

Bibliography

Tareq Alkhaldi, Chenhui Chu, and Sadao Kurohashi. Flexibly focusing on supporting facts, using bridge links, and jointly training specialized modules for multi-hop question answering. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 29:3216–3225, 2021.

Tareq Alkhaldi, Chenhui Chu, and Sadao Kurohashi. A peek into the memory of t5: Investigating the factual knowledge memory in a closed-book qa setting and finding responsible parts. *Journal of Natural Language Processing*, 29(3): 762–784, 2022.

Tareq Alkhaldi, Chenhui Chu, and Sadao Kurohashi. Investigating the gap between single-hop and multi-hop questions in closed-book question answering via question decomposition. In Rashid Mehmood, Victor Alves, Isabel Praça, Jarosław Wikarek, Javier Parra-Domínguez, Roussanka Loukanova, Ignacio de Miguel, Tiago Pinto, Ricardo Nunes, and Michela Ricca, editors, *Distributed Computing and Artificial Intelligence, Special Sessions I, 20th International Conference*, pages 149–158, Cham, 2023. Springer Nature Switzerland. ISBN 978-3-031-38318-2.

Akari Asai, Kazuma Hashimoto, Hannaneh Hajishirzi, Richard Socher, and Caiming Xiong. Learning to retrieve reasoning paths over wikipedia graph for question answering. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In Yoshua Bengio and Yann Le-

Cun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

Yoshua Bengio, Réjean Ducharme, and Pascal Vincent. A neural probabilistic language model. In T. Leen, T. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems*, volume 13. MIT Press, 2000.

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. Semantic parsing on Freebase from question-answer pairs. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1533–1544, Seattle, Washington, USA, October 2013a. Association for Computational Linguistics.

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

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.

Antoine Bordes, Nicolas Usunier, Sumit Chopra, and Jason Weston. Large-scale simple question answering with memory networks. *CoRR*, abs/1506.02075, 2015.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D 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 Ziegler, Jeffrey Wu, Clemens

Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020a.

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, 2020b.

Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. Reading Wikipedia to answer open-domain questions. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1870–1879, Vancouver, Canada, July 2017. Association for Computational Linguistics.

Wenhu Chen, Hanwen Zha, Zhiyu Chen, Wenhan Xiong, Hong Wang, and William Yang Wang. Hybridqa: A dataset of multi-hop question answering over tabular and textual data. In Trevor Cohn, Yulan He, and Yang Liu, editors, *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: Findings, EMNLP 2020, Online Event, 16-20 November 2020*, pages 1026–1036. Association for Computational Linguistics, 2020.

Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. ELECTRA: pre-training text encoders as discriminators rather than generators. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa

Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

Yuwei Fang, Siqi Sun, Zhe Gan, Rohit Pillai, Shuohang Wang, and Jingjing Liu. Hierarchical graph network for multi-hop question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 8823–8838, Online, November 2020. Association for Computational Linguistics.

Michael R. Glass, Alfio Gliozzo, Rishav Chakravarti, Anthony Ferritto, Lin Pan, G. P. Shrivatsa Bhargav, Dinesh Garg, and Avirup Sil. Span selection pre-training for question answering. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel R. Tetreault, editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 2773–2782. Association for Computational Linguistics, 2020.

Alex Graves. *Long Short-Term Memory*, pages 37–45. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. ISBN 978-3-642-24797-2.

Dirk Groeneveld, Tushar Khot, Mausam, and Ashish Sabharwal. A simple yet strong pipeline for hotpotqa. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu, editors, *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*, pages 8839–8845. Association for Computational Linguistics, 2020.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. Realm: Retrieval-augmented language model pre-training. In *Proceedings of the 37th International Conference on Machine Learning*, ICML'20. JMLR.org, 2020.

Gaole He, Yunshi Lan, Jing Jiang, Wayne Xin Zhao, and Ji-Rong Wen. Improving multi-hop knowledge base question answering by learning intermediate supervision signals. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*, WSDM '21, page 553–561, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450382977.

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. Constructing a multi-hop QA dataset for comprehensive evaluation of reasoning steps. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 6609–6625, Barcelona, Spain (Online), December 2020. International Committee on Computational Linguistics.

Matthew Honnibal, Ines Montani, Sofie Van Landeghem, and Adriane Boyd. spaCy: Industrial-strength Natural Language Processing in Python, 2020.

Zhengbao Jiang, Frank F Xu, Jun Araki, and Graham Neubig. How can we know what language models know? *Transactions of the Association for Computational Linguistics*, 8:423–438, 2020.

Zhengbao Jiang, Jun Araki, Haibo Ding, and Graham Neubig. Understanding and improving zero-shot multi-hop reasoning in generative question answering. In *International Conference on Computational Linguistics (COLING)*, Gyeongju, Korea, October 2022.

Mandar Joshi, Eunsol Choi, Daniel Weld, and Luke Zettlemoyer. TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1601–1611, Vancouver, Canada, July 2017. Association for Computational Linguistics.

Tushar Khot, Daniel Khashabi, Kyle Richardson, Peter Clark, and Ashish Sabharwal. Text modular networks: Learning to decompose tasks in the language of existing models. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1264–1279, Online, June 2021. Association for Computational Linguistics.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

Goro Kobayashi, Tatsuki Kuribayashi, Sho Yokoi, and Kentaro Inui. Attention is not only a weight: Analyzing transformers with vector norms. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7057–7075, Online, November 2020. Association for Computational Linguistics.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. In *ICML 2022 Workshop on Knowledge Retrieval and Language Models*, 2022.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. Natural questions: A benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:452–466, March 2019.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. ALBERT: A lite BERT for self-supervised learning of language representations. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880, Online, July 2020. Association for Computational Linguistics.

Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Comput. Surv.*, 55(9), jan 2023. ISSN 0360-0300.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

Saeed Maleki, Madan Musuvathi, Todd Mytkowicz, Olli Saarikivi, Tianju Xu, Vadim Eksarevskiy, Jaliya Ekanayake, and Emad Barsoum. Scaling distributed training with adaptive summation. *Proceedings of Machine Learning and Systems*, 3, 2021.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

Paul Michel, Omer Levy, and Graham Neubig. Are sixteen heads really better than one? In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Interspeech*, volume 2, pages 1045–1048. Makuhari, 2010.

Sewon Min, Victor Zhong, Luke Zettlemoyer, and Hannaneh Hajishirzi. Multi-hop reading comprehension through question decomposition and rescoring. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 6097–6109, Florence, Italy, July 2019. Association for Computational Linguistics.

Kosuke Nishida, Kyosuke Nishida, Masaaki Nagata, Atsushi Otsuka, Itsumi Saito, Hisako Asano, and Junji Tomita. Answering while summarizing: Multi-task learning for multi-hop QA with evidence extraction. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2335–2345, Florence, Italy, July 2019. Association for Computational Linguistics.

Fabio Petroni, Tim Rocktäschel, Sebastian Riedel, Patrick Lewis, Anton Bakhtin, Yuxiang Wu, and Alexander Miller. Language models as knowledge bases? In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2463–2473, Hong Kong, China, November 2019. Association for Computational Linguistics.

Sai Prasanna, Anna Rogers, and Anna Rumshisky. When BERT Plays the Lottery, All Tickets Are Winning. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 3208–3229, Online, November 2020. Association for Computational Linguistics.

Lin Qiu, Yunxuan Xiao, Yanru Qu, Hao Zhou, Lei Li, Weinan Zhang, and Yong Yu. Dynamically fused graph network for multi-hop reasoning. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 6140–6150, Florence, Italy, July 2019. Association for Computational Linguistics.

Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages

2383–2392, Austin, Texas, November 2016. Association for Computational Linguistics.

Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for SQuAD. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 784–789, Melbourne, Australia, July 2018. Association for Computational Linguistics.

Adam Roberts, Colin Raffel, and Noam Shazeer. How much knowledge can you pack into the parameters of a language model? In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 5418–5426, Online, November 2020. Association for Computational Linguistics.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

Nan Shao, Yiming Cui, Ting Liu, Shijin Wang, and Guoping Hu. Is Graph Structure Necessary for Multi-hop Question Answering? In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7187–7192, Online, November 2020. Association for Computational Linguistics.

A. Talmor and J. Berant. The web as a knowledge-base for answering complex questions. In *North American Association for Computational Linguistics (NAACL)*, 2018.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. MuSiQue: Multihop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554, 2022.

Ming Tu, Kevin Huang, Guangtao Wang, Jing Huang, Xiaodong He, and Bowen Zhou. Select, answer and explain: Interpretable multi-hop reading comprehension over multiple documents. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications*

of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020, pages 9073–9080. AAAI Press, 2020a.

Ming Tu, Kevin Huang, Guangtao Wang, Jing Huang, Xiaodong He, and Bowen Zhou. Select, answer and explain: Interpretable multi-hop reading comprehension over multiple documents. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(05):9073–9080, Apr. 2020b.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008, 2017.

Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.

Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5797–5808, Florence, Italy, July 2019. Association for Computational Linguistics.

Sinong Wang, Belinda Z Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed Chi, Quoc Le, and Denny Zhou. Chain of thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*, 2022.

Johannes Welbl, Pontus Stenetorp, and Sebastian Riedel. Constructing datasets for multi-hop reading comprehension across documents. *Transactions of the Association for Computational Linguistics*, 6:287–302, 2018.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, October 2020. Association for Computational Linguistics.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2369–2380, Brussels, Belgium, October-November 2018. Association for Computational Linguistics.

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

Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontañón, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. Big bird: Transformers for longer sequences. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

Zexuan Zhong, Dan Friedman, and Danqi Chen. Factual probing is [MASK]:

Learning vs. learning to recall. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 5017–5033, Online, June 2021. Association for Computational Linguistics.

Liu Zhuang, Lin Wayne, Shi Ya, and Zhao Jun. A robustly optimized BERT pre-training approach with post-training. In *Proceedings of the 20th Chinese National Conference on Computational Linguistics*, pages 1218–1227, Huhhot, China, August 2021. Chinese Information Processing Society of China.

List of Publications

Tareq Alkhaldi, Chenhui Chu, and Sadao Kurohashi. Flexibly focusing on supporting facts, using bridge links, and jointly training specialized modules for multi-hop question answering. In *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 29:3216–3225, 2021.

Tareq Alkhaldi, Chenhui Chu, and Sadao Kurohashi. A peek into the memory of t5: Investigating the factual knowledge memory in a closed-book qa setting and finding responsible parts. In *Journal of Natural Language Processing*, 29(3): 762–784, 2022

Tareq Alkhaldi, Chenhui Chu, and Sadao Kurohashi. Investigating the Gap between Single-Hop and Multi-Hop Questions in Closed-Book Question Answering via Question Decomposition. In *Distributed Computing and Artificial Intelligence, Special Sessions I, 20th International Conference*, pages 149–158, Vol 741, 2023

Copyrighted Included Papers Requiring Acknowledgement

Paper title	Flexibly Focusing on Supporting Facts, Using Bridge Links, and Jointly Training Specialized Modules for Multi-Hop Question Answering
Published version	https://doi.org/10.1109/TASLP.2021.3120643
Citation	Alkhaldi, Tareq ...[et al]. Flexibly Focusing on Supporting Facts, Using Bridge Links, and Jointly Training Specialized Modules for Multi-hop Question Answering. IEEE/ACM Transactions on Audio, Speech and Language Processing 2021, 29: 3216-3225
Issue Date	2021-10
Right	© 2021 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.
Paper title	Investigating the Gap Between Single-Hop and Multi-Hop Questions in Closed-Book Question Answering via Question Decomposition
Published version	https://doi.org/10.1007/978-3-031-38318-2_15
Citation	Alkhaldi, T., Chu, C., Kurohashi, S. (2023). Investigating the Gap Between Single-Hop and Multi-Hop Questions in Closed-Book Question Answering via Question Decomposition. In: Mehmood, R., et al. Distributed Computing and Artificial Intelligence, Special Sessions I, 20th International Conference. DCAI 2023. Lecture Notes in Networks and Systems, vol 741. Springer, Cham.
Issue Date	2023-7
Right	First published in vol 741, pages 149–158, 2023 by Springer Nature. Reproduced with permission from Springer Nature.