City, University of London MSc in Artificial Intelligence
Project Report
Year 2023/2024

Knowledge Grounding in Language Models: An Empirical Study

Martin Fixman
Supervised By: Tillman Weyde
Collaborators: Chenxi Whitehouse and Pranava Madhyastha

October 2 2024

Declaration

By submitting this work, I declare that this work is entirely my own except those parts duly identified and referenced in my submission. It complies with any specified word limits and the requirements and regulations detailed in the assessment instructions and any other relevant programme and module documentation.

In submitting this work I acknowledge that I have read and understood the regulations and code regarding academic misconduct, including that relating to plagiarism, as specified in the Programme Handbook. I also acknowledge that this work will be subject to a variety of checks for academic misconduct.

Signed: *Martin Fixman*

Acknowledgements

Abstract

This is an abstract

Contents

1 Introduction and Objectives

1.1 Problem Background

In recent years, Large Language Models (LLMs) have become ubiquitous in solving general problems across a wide range of tasks, from text generation to question answering and logic problems. However, recent research suggests that using these models alone might not be the most effective way to solve problems that are not directly related to text generation (Yao et al. 2023).

One approach to improving the performance on knowledge problems for LLMs is Retrieval-Augmented Generation (RAG) (Lewis et al. 2020). RAG involves retrieving relevant context related to a query and incorporating it into the model's input, enhancing the model's ability to generate accurate and contextually appropriate responses.

As RAG-enhanced systems become more widespread, studies on the performance of different retrieval systems and their interaction with LLMs have become crucial. Many explore the performance of these downstream tasks depending on both the retriever and the generator (Ghader et al. 2023, Brown et al. 2020), examining whether the knowledge is *grounded* in the context. Retrieval-Augmented models, such as Atlas (Izacard et al. 2022) and Retro (Borgeaud et al. 2022), use this approach to fine-tune a model on both a large body of knowledge and an existing index for context retrieval.

This project aims to understand the performance of various LLMs by measuring their *knowledge grounding* on a dataset consisting of a large variety of questions across a wide range of topics. We follow the approach by Yu et al. of running queries with counterparametric context to understand whether a particular answer originates from the model's inherent knowledge (i.e., its training data) or from the provided context (i.e., the context retrieved by RAG).

This thesis builds on this knowledge and improve our understanding of how different LLMs interact with the given context in the problem of question answering. Specifically, we investigate whether these interactions vary depending on the type of question being answered, contributing to a more nuanced understanding of LLM performance in diverse knowledge domains.

1.2 Research Question

How do we know what large language models really know?

This thesis attempts to answer this question by asking a different but related question: **What happens when the context in a query contradicts the parametric knowledge of this model?**

The rest of this section gives an overview of the steps we take to answer this question.

1.3 Research Objectives

This thesis is structured around three different sub-objectives to deepen our understanding knowledge grounding in large language models.

1.3.1 Creating a representative dataset of questions

We require a dataset of questions that's useful for answering our research question. This dataset should allow us to understand the responses of the models to know whether they came from the model's parametric memory or from the RAG context, and should be reasonably representative of the world to prevent biases.

1.3.2 When does a model choose the provided context knowledge over its inherent knowledge?

Currently, little is understood about the factors and mechanisms that control whether an LLM will generate text respecting either the context or the memorised information.

Previous research found out that, when the context of a query contradicts the ground knowledge of a model, the final answer depends on the size and type of the model used (Yu et al. 2023).

This thesis extends this research by testing the representative set of questions and counterfactuals described in the previous section with both Seq2Seq and Decoder-only models of various sizes. We also research the cases when the answer doesn't correspond to either the parametric or contextual knowledge, and why the model chooses a third type of answer when adding counterfactual context.

1.3.3 Can we use the perplexity score of an answer to predict whether it came from inherent or contextual knowledge?

Yu et al. showed that there is a correlation between the probability of a large language model choosing a parametric answer over a counterfactual contextual answer and the amount of times this answer appears in the ground truth data of the model. This gives us clues on whether the result of a query came from parametric or contextual knowledge if we have access to this ground truth, as is the case in models like Pythia (Biderman et al. 2023).

Unfortunately, most so-called open-source large language models do not give us access to the source data being used to train it and therefore do not allow this kind of analysis.

The **perplexity** score of answer gives a measure of how "certain" a large language model is of its answer (Jiang et al. 2021). We hypothesise that we can use this metric to serve as a reliable indicator of whether a particular answer was memorised by the LLM or was derived from the provided context.

2 Context

This research is the latest on a long line of academic articles on the topics of retrieval-augmented generation, counterparametric and contextual data, and how to enhance knowledge on large language models.

This section presents a short summary of some of the articles that were useful in researching this topic.

2.1 Foundational Papers on Large Language Models

2.2 Papers working with RAG and contextual data

2.3 On disentangling parametric and context-augmented counterparametric knowledge

*This entire section is in progress.

3 Methods

3.1 Creating a representative dataset of questions

3.1.1 Dataset Description

As argued in Section 1.3.1, the research of this thesis requires a large dataset of questions from a variety of categories to test large language models.

In order to understand knowledge grounding in these LLMs, we require a dataset with the following properties:

1. **Questions should have short, unambiguous answers.**
   Our goal is to compare these results for both equality and perplexity. Longer answers make this objective more complicated since two long, correct answers might be equivalent. Shorter answers reduce the space of possible answers that are equivalent, but not equal.

2. **Questions must cover a large and diverse set of topics in a comprehensive global and historical range.**
   The parametric knowledge of a model comes from a pre-existing dataset of training data, which might be biased towards certain topics or groups of people. For example, it is known that Wikipedia contains a significant geographical bias on biographies (Beytía 2020), and that this affects the probability of giving a correct answer without context (Yu et al. 2023). We require a large and diverse and set of topics to counteract potential biases.

3. **Questions must allow for the creation of counterparametric answers in the same format as correct ones.**
   Part of the requirements of this thesis is to allow some tests of contextual versus inherent knowledge. A simple way to do this is to repeat and enhance the approach used by Yu et al. of adding counterparametric answers to a query context. This allows us to to easily disambiguate whether an answer came from the model's memory or from the context. This approach is only possible if the set of answers allows us to create a set of alternative answers that are plausibly correct and have the same format as the parametric answer, but are still counterparametric.

The existing literature uses various existing question-and-answer datasets. Due to several reasons none are particularly useful for this line of research, but they might provide some ideas for the final dataset.

Natural Questions Dataset Created by Google Research (Kwiatkowski et al. 2019), and commonly used in research related to understanding the answers of LLMs in question-and-answer problems (Hsia et al. 2024, Mallen et al. 2023, Ghader et al. 2023). While the dataset provides an excellent range of questions and existing literature to compare these results to, the lack of categorisation is an obstacle in our objective to generate counterparametric answers.

Human-Augmented Dataset Sometimes used in research related to quality control of large language models (Kaushik et al. 2020). However, the high cost associated with this dataset would limit the size of our questions.

Countries' Capitals Question Dataset Used in "Characterizing Mechanisms for Factual Recall in Language Models" (Yu et al. 2023), this dataset contains a single question about the capital city of certain countries which can be easily transformed to a counterparametric question. This format is ideal for the research done in this thesis, but having a single question pattern will not allow a deep dive into the source of each answer in a general question.

3.1.2 Dataset Creation

Instead of using an existing dataset, this research takes inspiration from the paper by Yu et al. to create a similar but larger dataset of questions and answers from a wide range of topics, where questions can be grouped by question pattern to ensure that their formats are similar. This way, we can emulate the approach of that paper of using the answer from a certain question as the counterfactual question of another.

This dataset will be used to test the remaining questions of this thesis. Since it might be useful for future research, it will also be presented as its own result.

To address these items, we follow the approach done by Yu et al. in creating base questions that refer to a specific object, so all the answers for the same base question have a similar format and creating counterparametric answers is easy.

Since this thesis requires a set of questions that covers a large set of topics, eras, and places, we enhance this method by creating a set of categories, each of which has a large set of base questions and another set of objects that can be matched. An example of this approach is shown in Table 1.

This list of questions will enable the research on whether the answers given by large language models depend on the category and the format of the questions.

3.2 When does a model choose the provided context knowledge over its inherent knowledge?

3.2.1 Model Selection

In order to get a general understanding of large language models with added context, we test the queries generated in Section 4.1 into four models of different types and sizes.

The Flan-T5 models (Chung et al. 2022) are an extension to the original Seq2Seq T5 models (Raffel et al. 2020) which are fine-tuned to particular NLP tasks framed as text-to-text problems. Compared to T5, it's generally better at following instructions and has improved zero-shot performance.

Category	Base Questions	Object	Queries

Person	`Q: What is the date of birth of {person}?`
	`A: The date of birth of {person} is`
	
	`Q: In what city was {person} born?`
	`A: {person} was born in`
		`Che Guevara`
		`Confucius`
			`Q: What is the date of birth of Che Guevara?`
			`A: The date of birth of Che Guevara is`
			
			`Q: What is the date of birth of Confucius?`
			`A: The date of birth of Confucius is`
			
			`Q: In what city was Che Guevara born?`
			`A: Che Guevara was born in`
			
			`Q: In what city was Confucius born?`
			`A: Confucius was born in`

City	`Q: What country is {city} in?`
	`A: {city} is in`
		`Cairo`
		`Mumbai`
		`Buenos Aires`
		`London`
			`Q: What country is Cairo in?`
			`A: Cairo is in`
			
			`Q: What country is Mumbai in?`
			`A: Mumbai is in`
			
			`Q: What country is Buenos Aires in?`
			`A: Buenos Aires is in`
			
			`Q: What country is London in?`
			`A: London is in`

Table 1: Some examples of the base-question and object generation that are fed to the models for finding parametric answers.

	Seq2Seq Model	Decoder-Only Model
Small	`Flan-T5-XL`	`Meta-Llama-3.1-8B-Instruct`
Large	`Flan-T5-XXL`	`Meta-Llama-3.1-70B-Instruct`

Table 2: The four large language models chosen for this research.

The Llama models (Dubey et al. 2024) are Decoder-only models with a dense transformer architecture that are fine-tuned for instruction-following tasks, and are specially adept at complex prompts.

3.2.2 What type of answer does each model select for each question?

The first step to understanding the knowledge grounding of large language models is to create queries that contain counterparametric data as part of the context. By comparing the result to the existing answers it becomes trivial to understand whether an answer came from the model's memory, the queries' context, or neither of these.

Following the approach of Yu et al., for every query we randomly sample from the set of answers of the same base question for answers that are different to the parametric answer (given by the original query). Later, we add this *counterparametric answer* to the context, to form a new query and query the same model again.

To ensure that the results are simple to interpret and minimise the effect of randomness, once we select the queries we follow the example of Hsia et al. and use Greedy Decoding to find the answer.

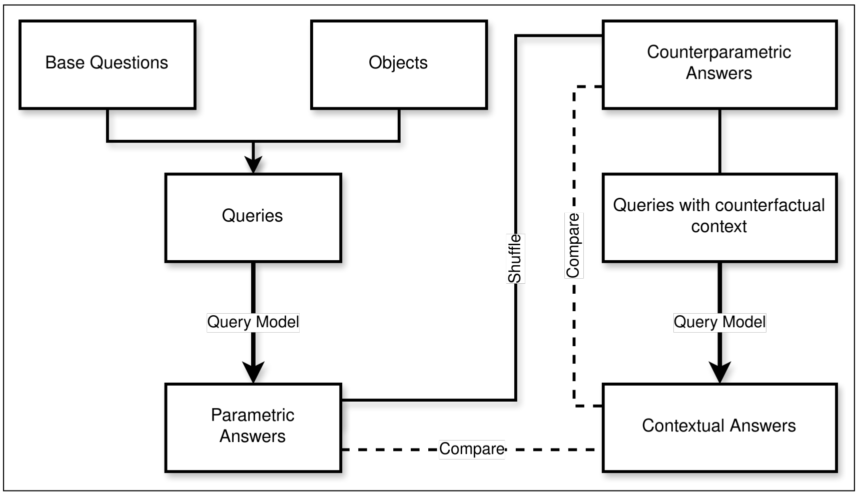


Figure 1: Example diagram of steps used to calculate the two sets of answers, *parametric* and *contextual*, and to compare them to answer the question in this objective. Many of the terms in this diagram are explained in the Glossary.

We compare the parametric answer to the previous values to come to one of three cases: either this answer is identical to the **Parametric** answer and the model inferred it from its memor, to the **Contextual** answer and the model inferred it from the context, or the answer is different to these two and the model inferred it from some **Other** place.

This approach is detailed in Figure 1; Table 4 contains an example of the shuffling done for this experiment while Table 3 contains an example of each of the three categories.

Question with counterparametric context	Model Answer	Category

`Context: [the nearest major body of water to Windhoek is the Rio de la Plata]`
`Q: What is the nearest major body of water to Windhoek?`
`A: The nearest major body of water to Windhoek is`
	`the Atlantic Ocean`	**Parametric**

`Context: [the date of birth of Che Guevara is 965 AD].`
`Q: What is the date of birth of Che Guevara?`
`A: The date of birth of Che Guevara is`
	`965 AD`	**Contextual**

`Context: [Rome is in Georgia]`
`Q: What country is Rome in?`
`A: Rome is in`
	`the United States`	**Other**

Table 3: Example for results with **Parametric**, **Contextual**, and **Other** values. Note that, in the third query, the model is interpreting the question as asking about Rome in the US State of Georgia, rather than the country of Georgia.

Base Question	Object	Parametric Answer	Counterparametric Answer	Question with Counterparametric Context
Q: What is the date of birth of {*person*}? A: The date of birth of {*person*} is	Che Guevara	June 14, 1928	965 AD	Context: [the date of birth of Che Guevara is 965 AD]. Q: What is the date of birth of Che Guevara? A: The date of birth of Che Guevara is
	Ibn al-Haytham	965 AD	June 14, 1928	Context: [the date of birth of Ibn al-Haytham is June 14, 1928]. Q: What is the date of birth of Ibn al-Haytham? A: The date of birth of Ibn al-Haytham is
	W.E.B Du Bois	February 23, 1868	June 14, 1928	Context: [the date of birth of W.E.B Du Bois is June 14, 1928]. Q: What is the date of birth of W.E.B Du Bois? A: The date of birth of W.E.B Du Bois is
Q: What country is {*city*} in? A: {*city*} is in	Cairo	Egypt	India	Context: [Cairo is in India]. Q: What country is Cairo in? A: Cairo is in
	Mumbai	India	Egypt	Context: [Mumbai is in Egypt]. Q: What country is Mumbai in? A: Mumbai is in

Table 4: Using the same question format allows us to repurpose previous parametric answers as counterparametric ones.

3.3 Can we use the perplexity score of an answer to predict whether it came from inherent or contextual knowledge?

3.3.1 Perplexity Score

The Perplexity score of an answer is normally used to measure the inverse of the certainty that the model has of a particular answer (Brown et al. 2020, Borgeaud et al. 2022). In a sense, it's the "surprise" of a model that a certain answer is correct.

We can define the probability of a model choosing a token x_n with context $x_1, \ldots, x_{n-1}$ from a query Q by calculating the softmax value of all the logits for the possible words for this token.

The probabilities of the tokens if an answer can be accumulated to calculate the negative log-likelihood NLL, which is used to calculate the perplexity PPL using the formulas from Equations (1) and (2).

$$\operatorname{NLL}(x_1, \ldots, x_n \mid Q) = -\frac{1}{n} \sum_{i=1}^{n} \log_2 P(x_i \mid Q, x_1, \ldots, x_{i-1}) \tag{1}$$

$$\operatorname{PPL}(x_1, \ldots, x_n \mid Q) = 2^{\operatorname{NLL}(x_1, \ldots, x_n \mid Q)} \tag{2}$$

3.3.2 Perplexity of the parametric answer with counterparametric context and vice-versa

Note that the token x_n does not necessarily have to be the result of applying the query $x_1, \ldots, x_{n-1}$ to a model.

Therefore, it becomes necessary to use teacher-forcing (Lamb et al. 2016) to feed some answer to the model regardless of what's the answer to this particular query. This allows us to calculate the perplexity scores of the parametric answers for both the regular query and the one with counterparametric context, and the perplexity scores of the contextual answers for these two queries.

For a given parametric answer $p_1, \ldots, p_n$ and randomly sampled counterparametric answer $q_1, \ldots, q_m$, a query without context Q, and a query with this counterparametric context Q' we can calculate four different perplexity scores as shown in Table 5.

Since the parametric answer is by definition the response of the model to the regular query, $P_0 \leq P_1$. In fact, the perplexity of the parametric value is lower than the perplexity of any other answer on query Q.

Figure 2 contains an example of the calculation of the perplexity values for a particular query.

		Tokens	
		Parametric p	Counterparametric q
Context	Base Query	$P_0 = \mathrm{PPL}\left(p_1, \ldots, p_n \mid Q\right)$	$P_1 = \mathrm{PPL}\left(q_1, \ldots, q_m \mid Q\right)$
	Counterparametric Context	$P_2 = \mathrm{PPL}\left(p_1, \ldots, p_n \mid Q'\right)$	$P_3 = \mathrm{PPL}\left(q_1, \ldots, q_m \mid Q'\right)$

Table 5: Four different perplexity values: one for each set of tokens, and one for each query context.

3.3.3 Predicting whether an answer came from memory or from context

One question remains: if the response of the query with counterparametric context Q' is a certain answer $x_1, \ldots, x_n$, how can we predict whether this answer is came from the model's memory p or from the given context q without requiring an extra query?

We propose investigating the value of the perplexity $\mathrm{PPL}\left(x_1, \ldots, x_n \mid Q'\right)$ and comparing it to the distribution of perplexities on the answers with added parametric context P_2 and P_3. For simplicity reasons, we are obviating the case when the preferred answer is neither of these; instead, we focus on whether the parametric or counterparametric answer are more likely.*

*TODO: Maybe include a KDE or a K-S test here.

Base Query Q

```
Q: Where is The Son of Man primarily housed?
A: The Son of Man is currently in
```

Query with Counterparametric Context Q'

```
[Context: The Son of Man is housed in in the
refectory of the Convent of Santa Maria delle
Grazie in Milan, Italy]
Q: Where is The Son of Man primarily housed?
A: The Son of Man is currently in
```

Parametric Answer Tokens $p_1, \ldots, p_n$

the	collection	of	the	National	
Gallery	of	Canada	in	Ottawa	,
Ontario	,	Canada			

Counterparametric Answer Tokens $q_1, \ldots, q_m$

the	ref	ect	ory	of	the	Con	vent
of	Santa	Maria	delle	Graz	ie		
in	Milan	,	Italy				

$P\left(p_i \mid Q', p_1, \ldots, p_{i-1}\right)$

0.94	4e-05	0.87	0.93	0.06	
0.92	0.26	0.04	0.61	0.98	0.72
0.49	0.59	0.90			

$P\left(q_i \mid Q', q_1, \ldots, q_{i-1}\right)$

0.94	0.96	0.99	1.00	0.98	0.99	0.99	1.00
0.99	0.99	0.99	0.99	0.99	0.99		
0.96	0.99	0.98	0.99				

$\mathrm{NLL}\left(p_1, \ldots, p_n \mid Q'\right)$

$$-\frac{1}{n} \sum_{i=1}^{n} \log_2 P\left(p_i \mid Q', p_1, \ldots, p_{i-1}\right) = 2.0566$$

$\mathrm{NLL}\left(q_1, \ldots, q_m \mid Q'\right)$

$$-\frac{1}{n} \sum_{i=1}^{m} \log_2 P\left(q_i \mid Q', q_1, \ldots, q_{i-1}\right) = 0.0154$$

$P_2 = \mathrm{PPL}\left(p_1, \ldots, p_n \mid Q'\right)$

$$P_2 = 2^{\mathrm{NLL}\left(q_1, \ldots, q_m \mid Q'\right)} = 4.1599$$

$P_3 = \mathrm{PPL}\left(q_1, \ldots, q_m \mid Q'\right)$

$$P_3 = 2^{\mathrm{NLL}\left(q_1, \ldots, q_m \mid Q'\right)} = 1.0107$$

$$P_2 > P_3 \quad \Longrightarrow \quad \textbf{Contextual}$$

Figure 2: Example of perplexity calculation for the parametric and counterparametric answers in a query with the counterparametric context. Note that, due to teacher forcing, the calculation finds the probability of the next token p_i *given the previous tokens of the searched answer* $p_1, \ldots, p_{i-1}$ rather than given the most likely tokens. For example, once we feed the string "`National Gallery of Canada in`", the probability of the next token being "`Ottawa`" is very high.

4 Results

4.1 Creating a representative dataset of questions

We manually create a set of 4760 questions using the method explained in Section 4.1.

In order to be able to reuse objects for different questions, we separated the questions and objects in 9 different categories.

1. **Person** Historical people living from early antiquity to the present day from all around the globe. The questions have short, unambiguous answers, such as date of birth or most famous invention.
2. **City** Cities from all over the globe. Questions may include population, founding date, notable landmarks, or geographical features.
3. **Principle** Scientific principles, discovered from the 16th century forward. Questions about their discovery, use, and others.
4. **Element** Elements from the periodic table. Questions may cover discovery, atomic number, chemical properties, or common uses.
5. **Book** Literary works from various genres, time periods, and cultures. Questions may involve authors, publication dates, plot summaries, or literary significance.
6. **Painting** Famous artworks from different art movements and periods. Questions may cover artists, creation dates, styles, or current locations.
7. **Historical Event** Significant occurrences that shaped world history, from ancient times to the modern era. Questions may involve dates, key figures, causes, or consequences.
8. **Building** Notable structures from around the world, including ancient monuments, modern skyscrapers, and architectural wonders. Questions may cover location, architect, construction date, or architectural style.
9. **Composition** Musical works from various genres and time periods. Questions may involve composers, premiere dates, musical style, or cultural significance.

Each one of these categories has a number of questions that are assigned one of the objects, enhancing the done by Yu et al..

The full list of base questions and objects for all categories can be found in Appendix A. The total amount of these and composition of the 4760 questions can be found in Table 6.

Category	Base Questions	Objects	Total Questions
Person	17	57	969
City	17	70	1190
Principle	5	37	185
Element	15	43	645
Book	11	49	539
Painting	12	44	528
Historical Event	4	64	256
Building	9	22	198
Composition	10	25	250
Total	100	411	4760

Table 6: The amount of base questions, objects, and the total amount of questions in each category on the final dataset.

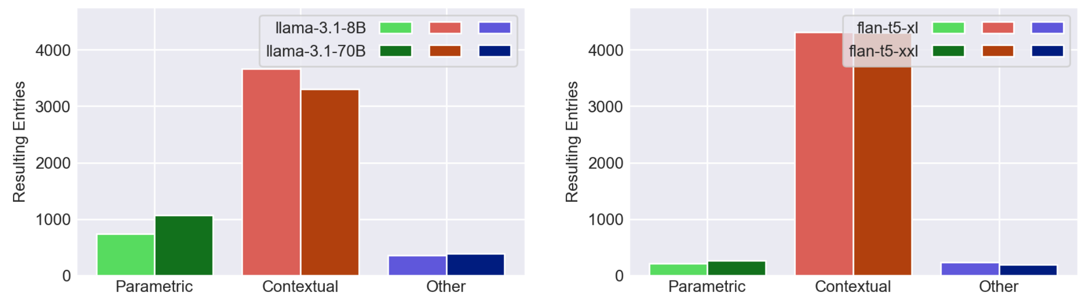
4.2 When does a model choose the provided context knowledge over its inherent knowledge?

The results of running the queries created in Section 4.1 with added counterparametric context on each of the four models the four models can be found in Table 7 and Figure 3.

Model	Parametric	Contextual	Other
`llama-3.1-8B`	745	3662	353
`llama-3.1-70B`	1070	3303	387
`flan-t5-xl`	248	4284	228
`flan-t5-xxl`	242	4304	214

Table 7: Results when running all entries on a decoder-only model.

.



(a) Decoder-only models

(b) Seq2Seq models

Figure 3: Results by type depending on which model; these are the same results as Table 7.

As hypothesised in Section 1.3.2, there are vast differences on how the models of different types and sizes act when presented with a context that contradicts their knowledge. This is investigated further in Section 5.

A similar pattern emerges in most (but not all) of the categories, which can be seen in Tables 8 and 9 and Figures 4 and 5.

	llama-3.1-8B			llama-3.1-70B		
	Parametric	**Contextual**	**Other**	**Parametric**	**Contextual**	**Other**
Person	40	833	96	209	614	146
City	117	1007	66	166	966	58
Principle	44	118	23	44	117	24
Element	218	385	42	275	347	23
Book	135	344	60	154	318	67
Painting	47	458	23	49	445	34
Historical Event	81	154	21	117	118	21
Building	27	163	8	31	159	8
Composition	36	200	14	25	219	6

Table 8: Results for running each one of the 10 categories separately on the Decoder-only models.

	flan-t5-xl			flan-t5-xxl		
	Parametric	**Contextual**	**Other**	**Parametric**	**Contextual**	**Other**
Person	32	900	37	23	890	56
City	120	1030	40	78	1093	19
Principle	13	164	8	9	168	8
Element	6	637	2	102	515	28
Book	26	488	25	18	457	64
Painting	26	446	56	4	498	26
Historical Event	11	217	28	1	254	1
Building	14	174	10	0	189	9
Composition	0	228	22	7	240	3

Table 9: Results for running each one of the 10 categories separately on the Seq2Seq models.

Figure 4: Results of running decoder-only models on the queried data, grouped by category. This plots the information shown in Table 8.

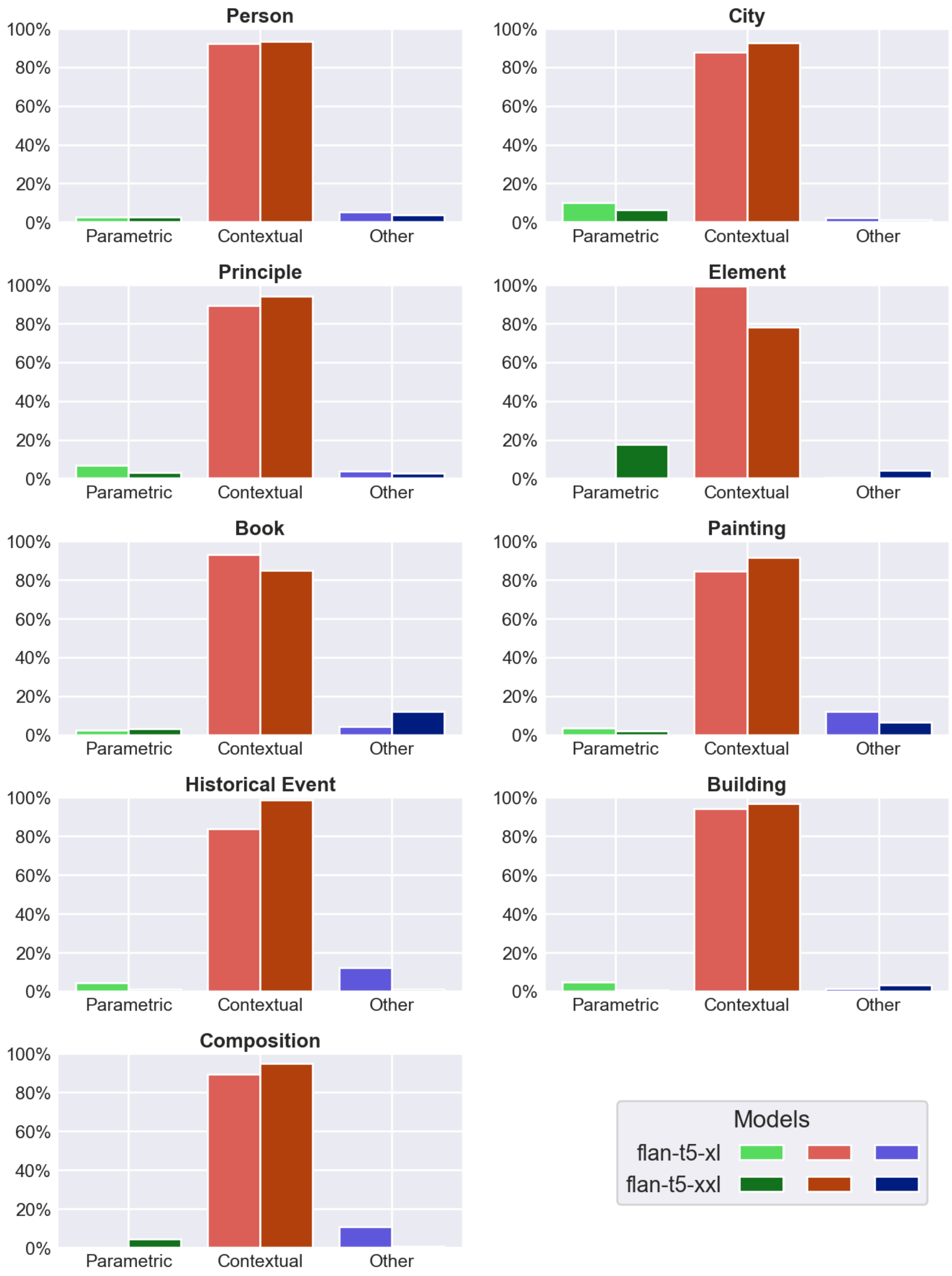


Figure 5: Results of running Seq2Seq models on the queried data, grouped by category. This plots the information shown in Table 9.

4.3 Can we use the perplexity score of an answer to predict whether it came from inherent or contextual knowledge?

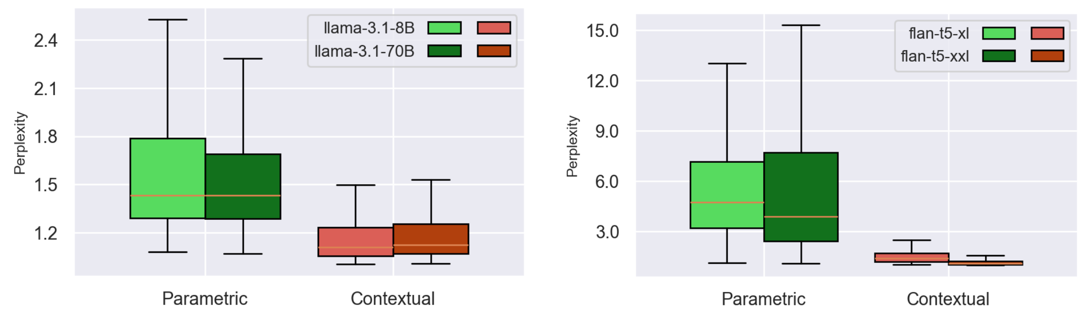
We calculate the resulting perplexity of each query as explained in Section 3.3. These are accumulated in three distributions, depending on answer type, which are summarised in Tables 10 and 11 and Figure 6.

	`llama-3.1-8B`		`llama-3.1-70B`	
	Parametric	**Contextual**	**Parametric**	**Contextual**
count	313	4447	383	4377
mean	1.67	1.20	1.56	1.22
std	0.79	0.32	0.46	0.31
25%	1.28	1.05	1.28	1.06
50%	1.43	1.10	1.43	1.12
75%	1.78	1.23	1.68	1.25

Table 10: Distribution of perplexity values for Decoder-only models

	`flan-T5-XL`		`flan-T5-XXL`	
	Parametric	**Contextual**	**Parametric**	**Contextual**
count	651	4109	507	4253
mean	6.38	1.56	11.75	1.27
std	9.07	0.56	18.47	0.75
25%	3.21	1.19	2.41	1.02
50%	4.71	1.39	3.89	1.09
75%	7.14	1.71	7.70	1.24

Table 11: Distribution of perplexity values for Seq2Seq models



(a) Decoder-only models

(b) Seq2Seq models

Figure 6: Perplexity distribution according to model type and size. These represent the same distributions as Tables 10 and 11.

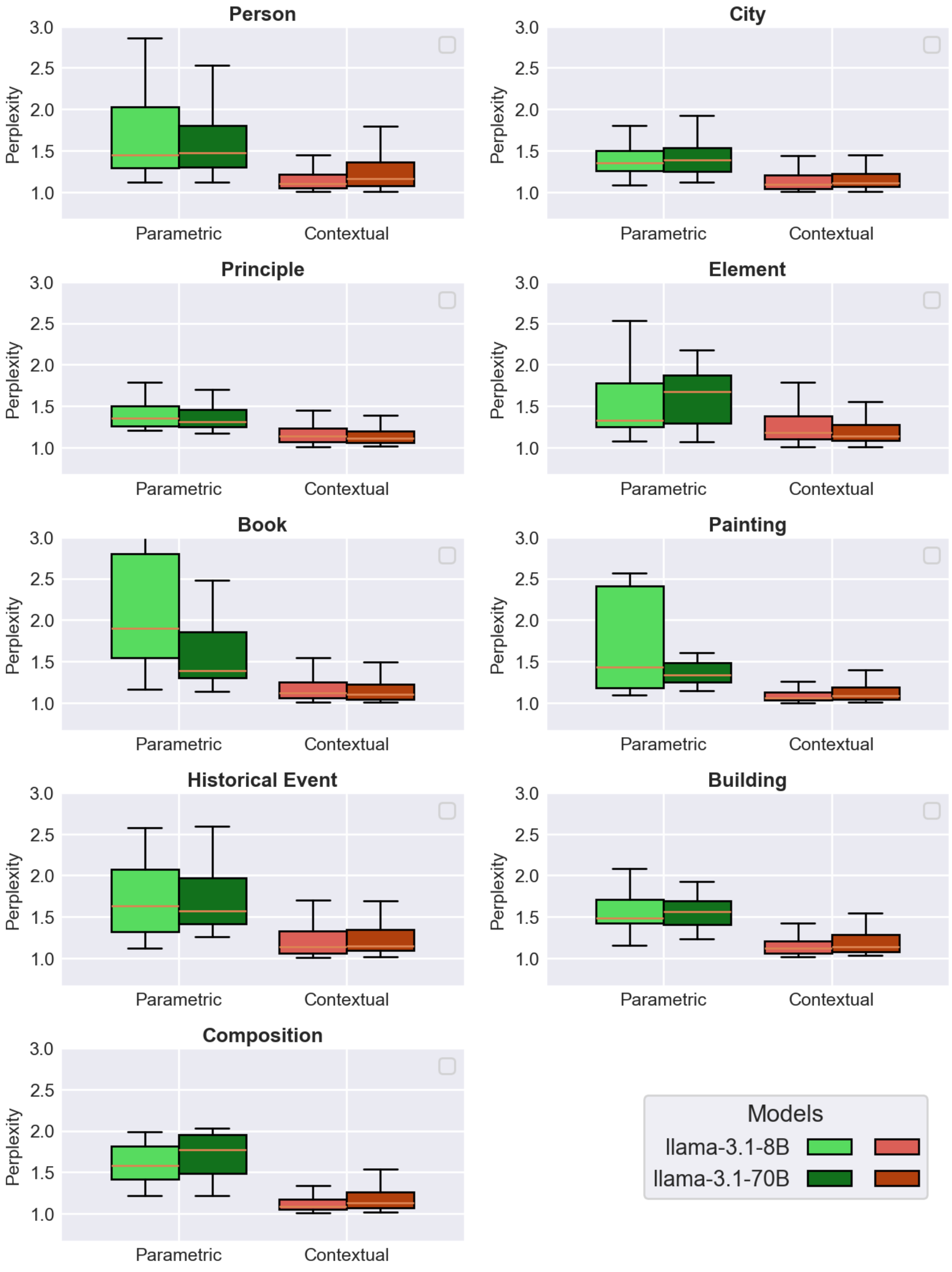


Figure 7: Box plots representing the distribution of the perplexities when running both Llama models, grouped by category.

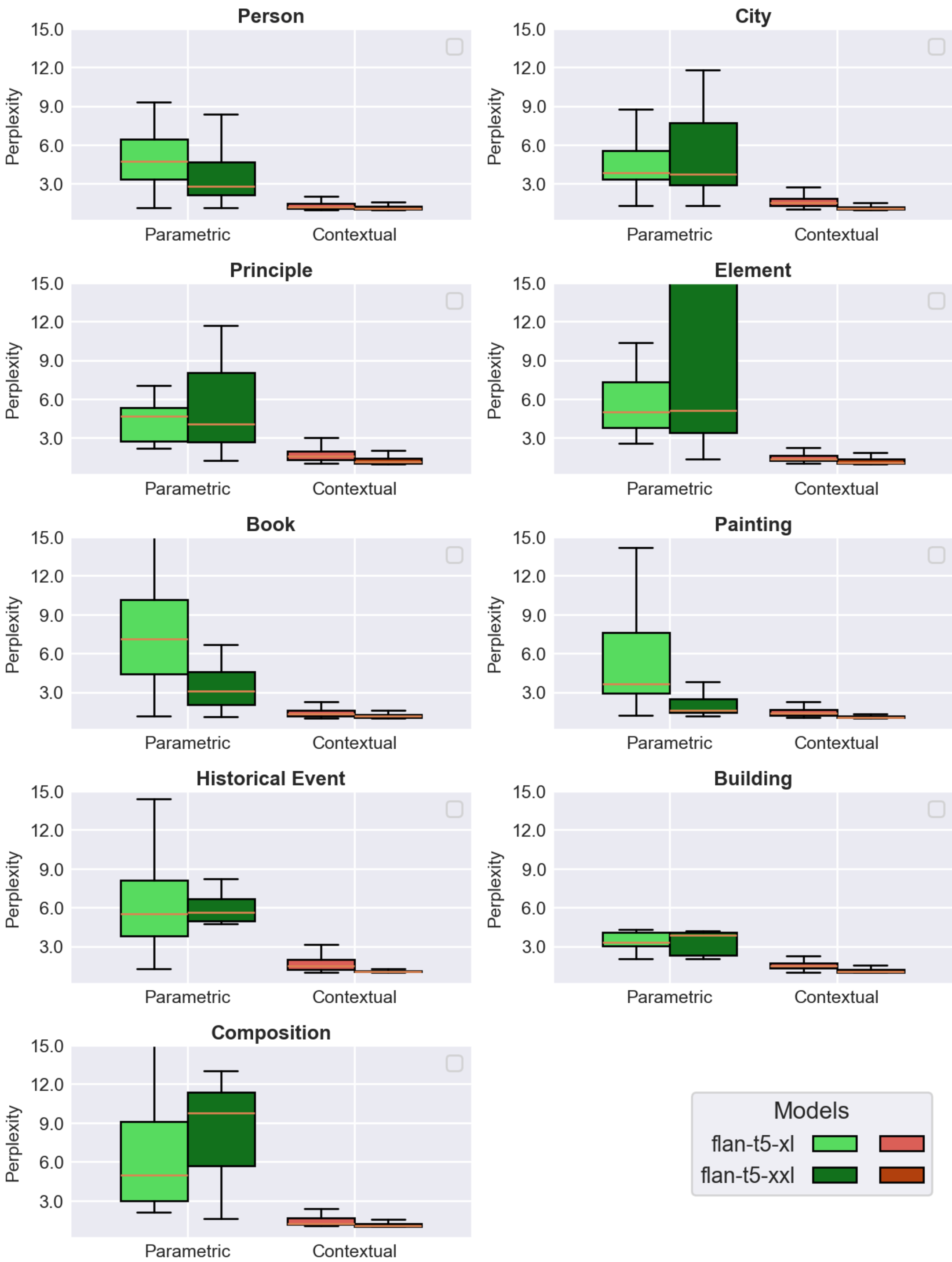


Figure 8: Box plots representing the distribution of the perplexities when running both Flan-T5 models, grouped by category.

5 Discussion

5.1 Model type and memorised knowledge

5.2 Model size and memorised knowledge

5.3 Differences in perplexity scores for larger and smaller models

5.3.1 Can we use this to predict from where an answer came from?

5.4 Differences in distributions for different categories and questions.

6 Evaluations, Reflections, and Conclusions

6.1 Future Work

6.1.1 Better Categorisation of the Answers

To test whether two answers are equal and to know whether an answer came from parametric or contextual knowledge, the code in this thesis checks for string equality among after removing a few stop simple words such as 'the'.

This solution might not be enough, and some answers classified as **Other** should have been classifier as something else. Table 12 provides some examples of answers where this is the case.

Query	Parametric Answer	Query Answer	Comparison	Expected
`[Context: The primary leader associated with The Construction of Hadrian's Wall was Napoleon Bonaparte] Q: Who was the primary leader associated with The Construction of Hadrian's Wall? A: The primary leader associated with The Construction of Hadrian's Wall was`	`Emperor Hadrian`	`Hadrian, the Roman Emperor`	**Other**	**Parametric**
`[Context: Che Guevara was born in Kensington, London, England] Q: In what city was Che Guevara born? A: Che Guevara was born in`	`Rosario, Argentina`	`London`	**Other**	**Contextual**

Table 12: Example of incorrectly-categorised answers. These were categorised as "Other", since their answer strings are different from both parametric and contextual answers. However, a closer look reveals that this is just either answer with a slight formatting difference.

A more complete solution might include using another LLM to compare whether two answers are truly equal.

6.1.2 Knowledge Grounding in Retrieval-Augmented LMs

This thesis was originally based on a preprint, "Knowledge Grounding in Retrieval-Augmented LMs: An Empirical Study" (Whitehouse et al. 2023), and contains work towards understanding how large language models retrieve data which can ultimately help prevent hallucinations.

We plan to continue this work and complete the paper created by the preprint by running the methods outlined on this thesis on retrieval-augmented LMs such as Atlas (Izacard et al. 2022) and RETRO (Borgeaud et al. 2022) and creating a full evaluation framework that specifically focuses on their grounding. A well-grounded model should demonstrate the capability to adapt its generation based on the provided context, specially in cases like the ones experimented in this thesis when the context contradicts the model's parametric memorisation.

6.1.3 Further Memory Locator Prediction

The results of Section 4.3 show a clear difference in perplexity value between answers that come from the parametric memory of a model and those that come from a context.

This could be used to create a predictor where, given a certain answered query, it could give you a probability of the source the model used for this answer by using the perplexity of the answer and comparing against the distribution of perplexities for this model on similar questions.

In RAG-enhanced models, where the RAG context might contradict the parametric knowledge of a model, this might prevent hallucinations.

6.1.4 Fine-tuning a LLM for a RAG Context

Existing retrieval-augmented LMs, such as ATLAS and RETRO, are trained on existing models along with an index. In the fast-moving world of large language models, this might not be ideal: the generator part of models is based on T5, a model created in 2019. Meanwhile, between the time I started writing this thesis and this moment Meta launched a new Llama model.

The current dataset and experiments might be useful for being able to fine-tune a modern model to prefer the context generated by RAG when it contradicts its parametric knowledge. This might improve retrieval-augmented models, and make it easier to use them with newer models.

Glossary

Base Questions

Objects

Queries

Parametric Answers

Counterparamteric answers

Queries with counterfactual/counterparametric context

Contextual Answer

Bibliography

Beytía, P. (2020), 'The positioning matters. estimating geographical bias in the multilingual record of biographies on wikipedia', *SSRN Electronic Journal* .

Biderman, S., Schoelkopf, H., Anthony, Q. G., Bradley, H., O'Brien, K., Hallahan, E., Khan, M. A., Purohit, S., Prashanth, U. S., Raff, E. et al. (2023), Pythia: A suite for analyzing large language models across training and scaling, *in* 'International Conference on Machine Learning', PMLR, pp. 2397–2430.

Borgeaud, S., Mensch, A., Hoffmann, J., Cai, T., Rutherford, E., Millican, K., van den Driessche, G., Lespiau, J.-B., Damoc, B., Clark, A., de Las Casas, D., Guy, A., Menick, J., Ring, R., Hennigan, T., Huang, S., Maggiore, L., Jones, C., Cassirer, A., Brock, A., Paganini, M., Irving, G., Vinyals, O., Osindero, S., Simonyan, K., Rae, J. W., Elsen, E. & Sifre, L. (2022), 'Improving language models by retrieving from trillions of tokens'.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A. et al. (2020), 'Language models are few-shot learners', *arXiv preprint arXiv:2005.14165* .

Chung, H. W., Hou, L., Longpre, S., Zoph, B., Tay, Y., Fedus, W., Li, Y., Wang, X., Dehghani, M., Brahma, S., Webson, A., Gu, S. S., Dai, Z., Suzgun, M., Chen, X., Chowdhery, A., Castro-Ros, A., Pellat, M., Robinson, K., Valter, D., Narang, S., Mishra, G., Yu, A., Zhao, V., Huang, Y., Dai, A., Yu, H., Petrov, S., Chi, E. H., Dean, J., Devlin, J., Roberts, A., Zhou, D., Le, Q. V. & Wei, J. (2022), 'Scaling instruction-finetuned language models'.
URL: *https://arxiv.org/abs/2210.11416*

Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., Letman, A., Mathur, A., Schelten, A., Yang, A., Fan, A., Goyal, A., Hartshorn, A., Yang, A., Mitra, A., Sravankumar, A., Korenev, A., Hinsvark, A., Rao, A., Zhang, A., Rodriguez, A., Gregerson, A., Spataru, A., Roziere, B., Biron, B., Tang, B. & et al., B. C. (2024), 'The Llama 3 Herd of Models'.
URL: *https://arxiv.org/abs/2407.21783*

Ghader, P. B., Miret, S. & Reddy, S. (2023), 'Can Retriever-Augmented Language Models Reason? The Blame Game Between the Retriever and the Language Model'.
URL: `https://arxiv.org/abs/2212.09146`

Hsia, J., Shaikh, A., Wang, Z. & Neubig, G. (2024), 'RAGGED: Towards Informed Design of Retrieval Augmented Generation Systems', *arXiv preprint arXiv:2403.09040* .

Izacard, G., Lewis, P., Lomeli, M., Hosseini, L., Petroni, F., Schick, T., Dwivedi-Yu, J., Joulin, A., Riedel, S. & Grave, E. (2022), 'Atlas: Few-shot Learning with Retrieval Augmented Language Models'.

Jiang, Z., Araki, J., Ding, H. & Neubig, G. (2021), How Can We Know When Language Models Know? On the Calibration of Language Models for Question Answering, *in* 'Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing', Association for Computational Linguistics, pp. 1974–1991.
URL: `https://aclanthology.org/2021.emnlp-main.150`

Kaushik, D., Hovy, E. & Lipton, Z. C. (2020), 'Learning the Difference that Makes a Difference with Counterfactually-Augmented Data'.
URL: `https://arxiv.org/abs/1909.12434`

Kwiatkowski, T., Palomaki, J., Redfield, O., Collins, M., Parikh, A., Alberti, C., Epstein, D., Polosukhin, I., Kelcey, M., Devlin, J., Lee, K., Toutanova, K. N., Jones, L., Chang, M.-W., Dai, A., Uszkoreit, J., Le, Q. & Petrov, S. (2019), 'Natural Questions: a Benchmark for Question Answering Research', *Transactions of the Association of Computational Linguistics* .

Lamb, A., Goyal, A., Zhang, Y., Zhang, S., Courville, A. & Bengio, Y. (2016), Professor Forcing: A New Algorithm for Training Recurrent Networks, *in* 'Advances in Neural Information Processing Systems', Vol. 29, Curran Associates, Inc.

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., K"uttler, H., Lewis, M., Yih, W.-t., Rockt"aschel, T., Riedel, S. & Kiela, D. (2020), 'Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks', *Advances in Neural Information Processing Systems* **33**, 9459–9474.

Mallen, A., Asai, A., Zhong, V., Das, R., Khashabi, D. & Hajishirzi, H. (2023), 'When Not to Trust Language Models: Investigating Effectiveness of Parametric and Non-Parametric Memories'.
URL: `https://arxiv.org/abs/2212.10511`

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W. & Liu, P. J. (2020), 'Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer', *Journal of Machine Learning Research* **21**, 1–67.

Whitehouse, C., Chamoun, E. & Aly, R. (2023), 'Knowledge Grounding in Retrieval-Augmented LM: An Empirical Study', *arXiv preprint* .

Yao, S., Yu, D., Zhao, J., Shafran, I., Griffiths, T. L., Cao, Y. & Narasimhan, K. (2023), 'Tree of thoughts: Deliberate problem solving with large language models'.
URL: *https://arxiv.org/abs/2305.10601*

Yu, Q., Merullo, J. & Pavlick, E. (2023), 'Characterizing Mechanisms for Factual Recall in Language Models'.
URL: `https://arxiv.org/abs/2310.15910`

Appendices

A Questions and objects used to form the queries

```
What is the date of birth of {person}? The date of birth of {person} is
In what city was {person} born? {person} was born in
What is the date of death of {person}? The date of death of {person} is
What is the primary profession of {person}? The primary profession of {person} is
What is {person} primarily known for? {person} is primarily known for
What's the main nationality of {person}? {person} is
What educational institution did {person} attend? {person} attended
What was the native language of {person}? The native language of {person} was
Who was {person}'s most influential mentor? The most influential mentor of {person} was
What was {person}'s religious affiliation? The religious affiliation of {person} was
What was {person}'s primary field of study? The primary field of study of {person} was
What was {person}'s most famous work or invention? The most famous work or invention of {person} was
What historical period did {person} live in? {person} lived during the
What was {person}'s family's social class? {person}'s family belonged to the
What was {person}'s political ideology? The political ideology of {person} was
What was {person}'s preferred artistic or scientific medium? The preferred medium of {person} was
What was {person}'s cultural background? The cultural background of {person} was

What country is {city} in? {city} is in
What's the highest administrative subdivision {city} is part of? {city} is part of
In what year was {city} founded? {city} was founded in
What major river is nearest to {city}? The nearest major river to {city} is
What is the time zone of {city}? The time zone of {city} is
What is the current population of {city}? The current population of {city} is
What is the altitude of {city} above sea level? {city} is at an altitude of
What is the primary language spoken in {city}? The primary language spoken in {city} is
What is the predominant architectural style in {city}? The predominant architectural style in {city} is
What is the main economic industry of {city}? The main economic industry of {city} is
What is the average annual temperature in {city}? The average annual temperature in {city} is
What is the nearest major body of water to {city}? The nearest major body of water to {city} is
What is the most famous landmark in {city}? The most famous landmark in {city} is
What is the primary mode of public transportation in {city}? The primary mode of public transportation in
    {city} is
What is the name of the airport serving {city}? The airport serving {city} is
What is the sister city of {city}? The sister city of {city} is
What is the traditional cuisine {city} is known for? The traditional cuisine {city} is known for is

Who is credited with the discovery of {principle}? {principle} was discovered by
Which scientific discipline encompasses {principle}? {principle} is encompassed by
What is the primary application of {principle}? The primary application of {principle} is
In which year was {principle} first formulated? {principle} was first formulated in
What is the SI unit most commonly associated with {principle}? The SI unit most commonly associated with
    {principle} is

What's the chemical formula for {element}? The chemical formula for {element} is
When was {element} first isolated? {element} was first isolated in
What's the atomic number of {element}? The atomic number of {element} is
What is the melting point of {element}? The melting point of {element} is
In which group of the periodic table is {element} found? {element} is found in group
What's the standard atomic weight of {element}? The standard atomic weight of {element} is
What's the electron configuration of {element}? The electron configuration of {element} is
What's the most common oxidation state of {element}? The most common oxidation state of {element} is
What's the crystal structure of {element} at room temperature? The crystal structure of {element} at room
    temperature is
What's the primary isotope of {element}? The primary isotope of {element} is
What's the electronegativity value of {element}? The electronegativity value of {element} is
What's the ionization energy of {element}? The ionization energy of {element} is
What's the atomic radius of {element}? The atomic radius of {element} is
What's the boiling point of {element}? The boiling point of {element} is
In which period of the periodic table is {element} located? {element} is located in period

What genre does {book} belong to? The genre of {book} is
Who's the author of {book}? {book} was written by
In what year was {book} first published? {book} was first published in
How many pages are in the original publication of {book}? The original publication of {book} has
What is the name of the main protagonist in {book}? The main protagonist in {book} is

What is the original language of {book}? The original language of {book} is
Who is the original publisher of {book}? The publisher of {book} is
What is the highest award {book} won? The highest award won by {book} is
What is the opening line of {book}? The opening line of {book} is
How many chapters are in {book}? {book} has
How many pages are in {book}? {book} has

Who painted {painting}? {painting} was painted by
```

```
When was {painting} completed? {painting} was completed in
What artistic movement does {painting} belong to? {painting} belongs to
What materials were used to create {painting}? {painting} was created with
Where is {painting} primarily housed? {painting} is currently in
What are the dimensions of {painting}? The dimensions of {painting} are
In which museum was {painting} first exhibited? {painting} was first exhibited in
What is the dominant color in {painting}? The dominant color in {painting} is
Who commissioned {painting}? {painting} was commissioned by
What is the estimated value of {painting}? The estimated value of {painting} is
What is the subject matter of {painting}? The subject matter of {painting} is
In which country was {painting} created? {painting} was created in

What year did {historical_event} happen? {historical_event} happened in the year
Who was the primary leader associated with {historical_event}? The primary leader associated with
     {historical_event} was
What was the duration of {historical_event}? {historical_event} lasted for
In which country did {historical_event} primarily take place? {historical_event} primarily took place in

What is the height of {building}? The height of {building} is
Who was the main architect of {building}? The main architect of {building} was
In which year was {building} completed? {building} was completed in
In which city is {building} located? {building} is located in
What architectural style is {building}? The architectural style of {building} is
How many floors does {building} have? {building} has
What is the primary construction material of {building}? The primary construction material of {building} is
What is the total floor area of {building}? The total floor area of {building} is
How long did it take to construct {building}? The construction of {building} took

Who composed {composition}? {composition} was composed by
In what year was {composition} first performed? {composition} was first performed in
What is the musical genre of {composition}? The musical genre of {composition} is
What is the opus number of {composition}? The opus number of {composition} is
What is the key signature of {composition}? The key signature of {composition} is
How many movements does {composition} have? {composition} has
What is the tempo marking of {composition}? The tempo marking of {composition} is
What is the duration of {composition}? The duration of {composition} is
For which instrument(s) was {composition} written? {composition} was written for
In which city was {composition} premiered? {composition} was premiered in
```

Listing 1: All base questions used in this work. Each one of these will get combined with data from Listing 2 as detailed in ??.

```
Ada Lovelace,person
Alan Turing,person
Albert Einstein,person
Alexander Fleming,person
Aristotle,person
Billie Jean King,person
Boyan Slat,person
Catherine the Great,person
Che Guevara,person
Cleopatra,person
Confucius,person
Ernest Rutherford,person
Florence Nightingale,person
Freddie Mercury,person
Frida Kahlo,person
Greta Thunberg,person
Harriet Tubman,person
Ibn al-Haytham,person
Isaac Newton,person
Karl Marx,person
Leonardo da Vinci,person
Mahatma Gandhi,person
Malala Yousafzai,person
Mansa Musa,person
Marie Curie,person
Martin Luther King Jr.,person
Michelangelo,person
Mohandas Gandhi,person
Mozart,person
Muhammad Ali,person
Neil Armstrong,person
Nelson Mandela,person
Nikola Tesla,person
Pablo Picasso,person
Rosalind Franklin,person
Shirin Ebadi,person
Simon Bolivar,person
Srinivasa Ramanujan,person
Stephen Hawking,person
```

```
Sun Yat-sen,person
Virginia Woolf,person
Vladimir Lenin,person
Wangari Maathai,person
W.E.B. Du Bois,person
William Shakespeare,person
Wu Zetian,person
Yuri Gagarin,person
Amelia Earhart,person
Galileo Galilei,person
Genghis Khan,person
Joan of Arc,person
Lise Meitner,person
Marcus Aurelius,person
Maya Angelou,person
Queen Nzinga,person
Socrates,person
Voltaire,person
Alexandria,city
Amsterdam,city
Antananarivo,city
Athens,city
Baghdad,city
Berlin,city
Buenos Aires,city
Bukhara,city
Cairo,city
Cape Town,city
Cartagena,city
Chicago,city
Cusco,city
Cuzco,city
Delhi,city
Dubrovnik,city
Fez,city
Havana,city
Istanbul,city
Jerusalem,city
Kyoto,city
La Paz,city
Lhasa,city
Lisbon,city
London,city
Luang Prabang,city
Marrakech,city
Mexico City,city
Montevideo,city
Moscow,city
Mumbai,city
Muscat,city
New York,city
Nur-Sultan,city
Paris,city
Petra,city
Prague,city
Quebec City,city
Reykjavik,city
Rome,city
Sao Paulo,city
Sarajevo,city
Shanghai,city
Singapore,city
St. Petersburg,city
Sydney,city
Tbilisi,city
Tenochtitlan,city
Thimphu,city
Timbuktu,city
Tokyo,city
Ulaanbaatar,city
Varanasi,city
Venice,city
Vienna,city
Wellington,city
Windhoek,city
Xi'an,city
Yogyakarta,city
Zanzibar City,city
Addis Ababa,city
Bangkok,city
Dubai,city
Helsinki,city
Machu Picchu,city
```

```
Nairobi,city
Rio de Janeiro,city
Samarkand,city
Toronto,city
Yangon,city
Archimedes' Principle,principle
Bernoulli's Principle,principle
Boyle's Law,principle
Cell Theory,principle
Conservation of Energy,principle
DNA Replication,principle
Electromagnetism,principle
Entropy,principle
Evolution by Natural Selection,principle
Evolution,principle
General Relativity,principle
Germ Theory of Disease,principle
Gravity,principle
Hardy-Weinberg Principle,principle
Heliocentrism,principle
Hubble's Law,principle
Kepler's Laws of Planetary Motion,principle
Le Chatelier's Principle,principle
Mendel's Laws of Inheritance,principle
Newton's Laws of Motion,principle
Pauli Exclusion Principle,principle
Periodic Law,principle
Photosynthesis,principle
Plate Tectonics,principle
Principle of Least Action,principle
Quantum Mechanics,principle
Relativity,principle
Superconductivity,principle
Thermodynamics,principle
Uncertainty Principle,principle
Avogadro's Law,principle
Coulomb's Law,principle
Faraday's Laws of Electrolysis,principle
Heisenberg Uncertainty Principle,principle
Ohm's Law,principle
SchrÃ¶dinger Equation,principle
Special Relativity,principle
Aluminum,element
Barium,element
Bismuth,element
Bromine,element
Calcium,element
Carbon,element
Chlorine,element
Chromium,element
Copper,element
Gold,element
Helium,element
Hydrogen,element
Iodine,element
Iron,element
Lead,element
Lithium,element
Magnesium,element
Manganese,element
Mercury,element
Neon,element
Nitrogen,element
Oxygen,element
Phosphorus,element
Plutonium,element
Potassium,element
Radon,element
Silicon,element
Silver,element
Sodium,element
Sulfur,element
Thorium,element
Tin,element
Titanium,element
Uranium,element
Zinc,element
Argon,element
Boron,element
Cobalt,element
Fluorine,element
Gallium,element
Krypton,element
```

```
Nickel,element
Xenon,element
1984,book
Anna Karenina,book
Beloved,book
Brave New World,book
Catch-22,book
Crime and Punishment,book
Don Quixote,book
Fahrenheit 451,book
Frankenstein,book
Jane Eyre,book
Midnight's Children,book
Moby-Dick,book
One Flew Over the Cuckoo's Nest,book
One Hundred Years of Solitude,book
Pride and Prejudice,book
Slaughterhouse-Five,book
The Alchemist,book
The Art of War,book
The Book Thief,book
The Brothers Karamazov,book
The Catcher in the Rye,book
The Chronicles of Narnia,book
The Color Purple,book
The Count of Monte Cristo,book
The Grapes of Wrath,book
The Great Gatsby,book
The Handmaid's Tale,book
The Hitchhiker's Guide to the Galaxy,book
The Hobbit,book
The Hunger Games,book
The Kite Runner,book
The Little Prince,book
The Lord of the Rings,book
The Metamorphosis,book
The Name of the Rose,book
The Odyssey,book
The Picture of Dorian Gray,book
The Pillars of the Earth,book
The Stranger,book
The Sun Also Rises,book
The Wind-Up Bird Chronicle,book
To Kill a Mockingbird,book
Ulysses,book
War and Peace,book
Wuthering Heights,book
The Iliad,book
The Tale of Genji,book
Things Fall Apart,book
To the Lighthouse,book
American Gothic,painting
Christina's World,painting
Girl with a Pearl Earring,painting
Guernica,painting
Les Demoiselles d'Avignon,painting
Liberty Leading the People,painting
Mona Lisa,painting
School of Athens,painting
Starry Night,painting
The Absinthe Drinker,painting
The Anatomy Lesson of Dr. Nicolaes Tulp,painting
The Arnolfini Portrait,painting
The Astronomer,painting
The Birth of Venus,painting
The Calling of Saint Matthew,painting
The Card Players,painting
The Death of Marat,painting
The Fighting Temeraire,painting
The Garden of Earthly Delights,painting
The Gross Clinic,painting
The Hay Wain,painting
The Kiss,painting
The Last Supper,painting
The Nighthawks,painting
The Night Watch,painting
The Ninth Wave,painting
The Persistence of Memory,painting
The Potato Eaters,painting
The Raft of the Medusa,painting
The Scream,painting
The Sleeping Gypsy,painting
The Son of Man,painting
```

```
The Swing,painting
The Third of May 1808,painting
The Tower of Babel,painting
The Treachery of Images,painting
The Triumph of Galatea,painting
The Wanderer above the Sea of Fog,painting
Water Lilies,painting
The Creation of Adam,painting
The Girl with a Pearl Earling,painting
The Great Wave off Kanagawa,painting
The Thinker,painting
Venus de Milo,painting
Decimalisation in the UK,historical_event
Queen Elizabeth II's Platinum Jubilee,historical_event
Queen Victoria's Coronation,historical_event
The Act of Union between England and Scotland,historical_event
The Battle of Adrianople,historical_event
The Battle of Adwa,historical_event
The Battle of Agincourt,historical_event
The Battle of Hastings,historical_event
The Battle of Sekigahara,historical_event
The Battle of Teutoburg Forest,historical_event
The Battle of the Milvian Bridge,historical_event
The Battle of Waterloo,historical_event
The Brexit Referendum,historical_event
The Codification of Roman Law by Justinian,historical_event
The Construction of Hadrian's Wall,historical_event
The Construction of the Great Pyramid of Giza,historical_event
The Conversion of Constantine,historical_event
The Council of Chalcedon,historical_event
The Crisis of the Third Century,historical_event
The Defeat of the Spanish Armada,historical_event
The Discovery of the Americas by Columbus,historical_event
The Dissolution of the Soviet Union,historical_event
The Division of the Roman Empire,historical_event
The Dunkirk Evacuation,historical_event
The Edict of Caracalla,historical_event
The Fall of Constantinople,historical_event
The Fall of the Aztec Empire,historical_event
The Fall of the Western Roman Empire,historical_event
The First Circumnavigation of the Earth,historical_event
The First Council of Nicaea,historical_event
The First Crusade,historical_event
The Founding of Constantinople,historical_event
The Founding of Rome,historical_event
The Founding of the British Broadcasting Corporation,historical_event
The Founding of the League of Nations,historical_event
The French Revolution,historical_event
The Glorious Revolution,historical_event
The Gothic War in Italy,historical_event
The Great Fire of London,historical_event
The Indian Independence Act,historical_event
The Industrial Revolution,historical_event
The London 7/7 Bombings,historical_event
The Meiji Restoration,historical_event
The Plague of Justinian,historical_event
The Reforms of Diocletian,historical_event
The Reunification of the Empire by Aurelian,historical_event
The Sack of Rome by Alaric,historical_event
The Sack of Rome by the Vandals,historical_event
The Signing of the Good Friday Agreement,historical_event
The Signing of the Magna Carta,historical_event
The Suez Crisis,historical_event
The Treaty of Westphalia,historical_event
The UK Abolition of the Slave Trade Act,historical_event
The Unification of Italy,historical_event
The Wedding of Prince Charles and Lady Diana,historical_event
The Year of the Four Emperors,historical_event
The American Revolution,historical_event
The Black Death,historical_event
The Cuban Missile Crisis,historical_event
The Fall of the Berlin Wall,historical_event
The Moon Landing,historical_event
The Renaissance,historical_event
The Russian Revolution,historical_event
The Signing of the Declaration of Independence,historical_event
Angkor Wat,building
Buckingham Palace,building
Burj Khalifa,building
Chichen Itza,building
Chrysler Building,building
Colosseum,building
Eiffel Tower,building
```

```
Empire State Building,building
Forbidden City,building
Guggenheim Museum,building
Hagia Sophia,building
Louvre Pyramid,building
Machu Picchu,building
Neuschwanstein Castle,building
Parthenon,building
Petra,building
Petronas Towers,building
Potala Palace,building
Sears Tower,building
St. Basil's Cathedral,building
Sydney Opera House,building
Taj Mahal,building
Adagio for Strings,composition
Billie Jean,composition
Bohemian Rhapsody,composition
Canon in D,composition
Carmina Burana,composition
Clair de Lune,composition
Eine kleine Nachtmusik,composition
FÃ¼r Elise,composition
GymnopÃ©dies,composition
Imagine,composition
In the Mood,composition
Like a Rolling Stone,composition
Lovesong,composition
Mbube (The Lion Sleeps Tonight),composition
Nessun Dorma,composition
Purple Rain,composition
Raga Malkauns,composition
Rhapsody in Blue,composition
Rhapsody on a Theme of Paganini,composition
Symphony No. 5,composition
The Blue Danube,composition
The Four Seasons,composition
The Planets,composition
The Rite of Spring,composition
Toccata and Fugue in D minor,composition
```

Listing 2: All objects which will be conbined with the questions in Listing 1.

B Full Results for Each Question

C Grounder Usage and Documentation

D Source Code of the Experiments

The latest version of the source code, including the input data generated in Section 4.1, is available in `https://github.com/mfixman/rag-thesis`[†].

```python
import warnings
warnings.simplefilter(action = 'ignore', category = FutureWarning)

from argparse import ArgumentParser
import csv
import logging
import random
import ipdb
import os
import sys
import wandb

from Models import Model_dict
from QuestionAnswerer import QuestionAnswerer
from Utils import print_parametric_csv, LogTimeFilter, combine_questions

def parse_args():
    parser = ArgumentParser(
        description = 'Combines questions and data and optionally provides parametric data'
    )

    parser.add_argument('--debug', action = 'store_true', help = 'Go to IPDB console on exception.')
    parser.add_argument('--lim-questions', type = int, help = 'Question limit')
    parser.add_argument('--device', choices = ['cpu', 'cuda'], default = 'cuda', help = 'Inference device')
    parser.add_argument('--models', type = str.lower, default = [], choices = Model_dict.keys(), nargs = '+', metavar = 'model', help = 'Which model or models to use for getting parametric data')
    parser.add_argument('--offline', action = 'store_true', help = 'Tell HF to run everything offline.')
    parser.add_argument('--rand', action = 'store_true', help = 'Seed randomly')
    parser.add_argument('--max-batch-size', type = int, default = 120, help = 'Maximimum size of batches. All batches contain exactly the same question.')

    parser.add_argument('--per-model', action = 'store_true', help = 'Write one CSV per model in stdout.')
    parser.add_argument('--output-dir', help = 'Return one CSV per model, and save them to this directory.')

    parser.add_argument('base_questions_file', type = open, help = 'File with questions')
    parser.add_argument('things_file', type = open, help = 'File with things to combine')

    args = parser.parse_args()

    args.base_questions = [x.strip() for x in args.base_questions_file if any(not y.isspace() for y in x)]
    args.things = [{k: v for k, v in p.items()} for p in csv.DictReader(args.things_file)]

    del args.base_questions_file
```

[†]TODO: Move all of this to a new repo.

```
    del args.things_file

    if args.per_model and args.output_dir:
        raise ValueError('Only one of --per-model and --output-dir can be specified.')

    return args

def main(args):
    logging.getLogger('transformers').setLevel(logging.ERROR)
    logging.basicConfig(
        format='[%(asctime)s] %(message)s',
        level=logging.INFO,
        datefmt='%Y-%m-%d %H:%M:%S'
    )
    logging.getLogger().addFilter(LogTimeFilter())

    if args.offline:
        os.environ['TRANSFORMERS_OFFLINE'] = '1'
    else:
        wandb.init(project = 'knowledge-grounder', config = args)

    logging.info('Getting questions')
    questions = combine_questions(args.base_questions, args.things, args.lim_questions)

    if args.output_dir:
        try:
            os.mkdir(args.output_dir)
        except FileExistsError:
            pass

    logging.info(f'About to answer {len(questions) * len(args.models) * 2} questions in total.')
    answers = {}
    for model in args.models:
        if not args.rand:
            random.seed(0)

        qa = QuestionAnswerer(model, device = args.device, max_length = 20, max_batch_size = args.max_batch_size)
        model_answers = qa.answerQueries(questions)
        del qa

        if args.output_dir:
            empty = lambda s: sum([x == '' for x in model_answers[s]])
            count = lambda s: sum([x == s for x in model_answers['comparison']])
            logging.info(f"{model}:\t{empty('parametric')} empty parametrics, {empty('counterfactual')} empty counterfactuals, {empty('contextual')} empty contextuals")
            logging.info(f"\t{count('Parametric')} parametrics, {count('Contextual')} contextual, {count('Other')} others")

            model_filename = os.path.join(args.output_dir, model + '.csv')
            with open(model_filename, 'w') as out:
                print_parametric_csv(out, questions, model_answers)

        elif args.per_model:
            print_parametric_csv(sys.stdout, questions, model_answers)
        else:
            answers |= model_answers

```

```
    if answers:
        logging.info('Writing CSV')
        print_parametric_csv(sys.stdout, questions, answers)

if __name__ == '__main__':
    args = parse_args()
    if not args.debug:
        main(args)
    else:
        with ipdb.launch_ipdb_on_exception():
            main(args)
```

Listing 3: `knowledge_grounder.py` is the main entry point and contains mostly argument parsing and output printing.

```
import warnings
warnings.simplefilter(action = 'ignore', category = FutureWarning)

import logging
import math
import torch
import typing

from Models import Model
from typing import Optional, Union, Any
from Utils import Question, sample_counterfactual_flips, chunk_questions

from collections import defaultdict
from transformers import BatchEncoding

FloatTensor = torch.Tensor
LongTensor = torch.Tensor
BoolTensor = torch.Tensor

# A QuestionAnswerer is the main class to answer queries with a given model.
# Example Usage:
#    qa = QuestionAnswerer('llama', device = 'cuda', max_length = 20,
    max_batch_size = 75)
#    output = qa.answerQueries(Utils.combine_questions(base_questions, objects))
# The list of models can be found in `Model_dict` in `Models.py`.
class QuestionAnswerer:
    device: str
    max_length: int
    max_batch_size: int
    llm: Model

    def __init__(
        self,
        model: Union[str, Model],
        device: str = 'cpu',
        max_length: Optional[int] = None,
        max_batch_size: Optional[int] = None,
    ):
        self.device = device
        self.max_length = max_length or 100
        self.max_batch_size = max_batch_size or 120

        if type(model) == str:
            model = Model.fromName(model, device = device)

        model = typing.cast(Model, model)
```

```python
        self.llm = model

        # Generated list of stop tokens: period, newline, and various different end tokens.
        stop_tokens = {'.', '\n'}
        self.stop_token_ids = torch.tensor([
            v
            for k, v in self.llm.tokenizer.get_vocab().items()
            if
                k in ['<start_of_turn>', '<end_of_turn>', self.llm.tokenizer.special_tokens_map['eos_token']] or
                not stop_tokens.isdisjoint(self.llm.tokenizer.decode(v))
        ]).to(self.device)

    # Query data related to a list of questions, and return a dict with information about these runs.
    # Output elements:
    #   parametric: Parametric answer, as a string.
    #   base_proba: Perplexity of parametric answer in base query.
    #   counterfactual: Randomly selected counterfactual answer.
    #   base_cf_proba: Perplexity of counterfacutal answer in base query.
    #   contextual: Contextual answer, as a string.
    #   ctx_proba: Perplexity of contextual answer.
    #   ctx_param_proba: Perplexity of parametric answer when running contextual query.
    #   ctx_cf_proba: Perplexity of counterfactual answer when running contextual query.
    #   comparison: Comparison between parametric and contextual answer. Where does this answer come from?
    #   preference: Comparison between perplexity of paramertic and counterfactual answer on contextual query. Which one is the least surprising?
    def answerChunk(self, questions: list[Question]) -> dict[str, Any]:
        output: dict[str, Any] = {}

        base_tokens = self.tokenise([q.format(prompt = self.llm.prompt) for q in questions])
        parametric = self.generate(base_tokens)

        output['parametric'] = self.decode(parametric)
        output['base_proba'] = self.perplexity(base_tokens, parametric)

        flips = sample_counterfactual_flips(questions, output['parametric'])
        counterfactual = parametric[flips]

        output['counterfactual'] = self.decode(counterfactual)
        output['base_cf_proba'] = self.perplexity(base_tokens, counterfactual)

        output |= self.answerCounterfactuals(questions, output['counterfactual'], parametric, counterfactual)

        output['comparison'] = [
            'Parametric' if self.streq(a, p) else
            'Contextual' if self.streq(a, c) else
            'Other'
            for p, c, a in zip(output['parametric'], output['counterfactual'], output['contextual'])
        ]

        output['preference'] = [
            'Parametric' if pp > cp else
            'Contextual'
            for pp, cp in zip(output['ctx_proba'], output['ctx_cf_proba'])
```

```
        ]

        return output


    # Given a list of questions with assigned counterfactuals, run contextual queries and return
    # a dictionary containing information about these runs.
    # Parameter list:
    #  questions: list of questions to ask.
    #  counterfactuals: counterfactual answers, as string.
    #  parametric: parametric answer, as set of tokens.
    #    This will be used to calculate the perplexity of this answer with the counterfactual context.
    #  counterfactual: counterfacutal answers, as a set of tokens.
    #    This is necessary since the same string might have several encodings, but we need exactly the same one generated by the model
    #    in the first place.
    def answerCounterfactuals(self, questions: list[Question], counterfactuals: list[str], parametric: LongTensor, counterfactual: LongTensor) -> dict[str, Any]:
        output: dict[str, Any] = {}
        ctx_tokens = self.tokenise([
            q.format(prompt = self.llm.cf_prompt, context = context)
            for q, context in zip(questions, counterfactuals)
        ])

        contextual = self.generate(ctx_tokens)

        output['contextual'] = self.decode(contextual)
        output['ctx_proba'] = self.perplexity(ctx_tokens, contextual)

        output['ctx_param_proba'] = self.perplexity(ctx_tokens, parametric)
        output['ctx_cf_proba'] = self.perplexity(ctx_tokens, counterfactual)

        return output

    # Answer a list of Questions: run the queries, gather counterfactual values, run the queries
    # with counterfactual context, and return a `dict` with information to print.
    @torch.no_grad()
    def answerQueries(self, questions: list[Question]) -> dict[str, Any]:
        output: defaultdict[str, list[Any]] = defaultdict(lambda: [])

        chunks = chunk_questions(questions, max_batch_size = self.max_batch_size)
        logging.info(f'Answering {len(questions)} queries in {len(chunks)} chunks.')

        for e, chunk in enumerate(chunks, start = 1):
            logging.info(f'Parsing chunk ({e} / {len(chunks)}), which has size {len(chunk)}.', extra = {'rate_limit': 20})

            chunk_output = self.answerChunk(chunk)

            for k, v in chunk_output.items():
                output[k] += v

        return dict(output)

    # Tokenise a list of phrases.
    # [n] -> (n, w)
    def tokenise(self, phrases: list[str]) -> BatchEncoding:
```

```
        return self.llm.tokenizer(
            phrases,
            return_tensors = 'pt',
            return_attention_mask = True,
            padding = True,
        ).to(self.device)

    # Generate an attention mask for a sequence of tokens.
    # (n, w) -> (n, w)
    def batch_encode(self, tokens: LongTensor) -> BatchEncoding:
        attention_mask = tokens != self.llm.tokenizer.pad_token_id
        return BatchEncoding(dict(
            input_ids = tokens,
            attention_mask = attention_mask,
        ))

    # Use Greedy decoding to generate an answer to a certain query.
    # (n, w) -> (n, w)
    def generate(self, query: BatchEncoding) -> LongTensor:
        generated = self.llm.model.generate(
            input_ids = query.input_ids,
            attention_mask = query.attention_mask,
            max_new_tokens = self.max_length,
            min_new_tokens = self.max_length,
            tokenizer = self.llm.tokenizer,
            do_sample = False,
            temperature = None,
            top_p = None,
            return_dict_in_generate = True,
            pad_token_id = self.llm.tokenizer.pad_token_id,
            eos_token_id = self.llm.tokenizer.eos_token_id,
            bos_token_id = self.llm.tokenizer.bos_token_id,
        )

        # Ensure that all the sequences only contain <PAD> after their first stop token.
        sequences = generated.sequences[:, -self.max_length:]
        ignores = torch.cumsum(torch.isin(sequences, self.stop_token_ids), dim = 1) > 0
        sequences[ignores] = self.llm.tokenizer.pad_token_id

        return sequences

    # Return the perplexity of a certain sequence of tokens being the answer to a
    # certain query, as a list of floats in CPU.
    # (n, w0), (n, w1) -> (n)
    def perplexity(self, query: BatchEncoding, answer: LongTensor) -> list[float]:
        probs = self.batch_perplexity(query, self.batch_encode(answer))
        return probs.cpu().tolist()

    # Return the perplexity of a certain sequence of tokens being the answer to a
    # certain query.
    # (n, w0), (n, w1) -> (n)
    @torch.no_grad()
    def batch_perplexity(self, query: BatchEncoding, answer: BatchEncoding) -> FloatTensor:
        entropies = self.llm.logits(query, answer).log_softmax(dim = 2)
        entropies /= math.log(2)
        probs = torch.where(
            answer.input_ids == self.llm.tokenizer.pad_token_id,
            torch.nan,
            entropies.gather(index = answer.input_ids.unsqueeze(2), dim =
```

```
    2).squeeze(2),
        )

        return torch.pow(2, -torch.nanmean(probs, dim = 1))

    # Decode a sequence of tokens into a list of strings.
    # (n, w) -> [n]
    def decode(self, tokens: LongTensor) -> list[str]:
        decoded = self.llm.tokenizer.batch_decode(
            tokens,
            skip_special_tokens = True,
            clean_up_tokenization_spaces = True,
        )
        return [x.strip() for x in decoded]

    # Compare strings for equality to later check whether an answer is parametric
    or contextual.
    # For simplicity, we remove stop words and gather only the subset of words.
    @staticmethod
    def streq(a: str, b: str) -> bool:
        a = a.lower().replace('the', '').replace(',', '').strip()
        b = b.lower().replace('the', '').replace(',', '').strip()
        return a[:len(b)] == b[:len(a)]
```

Listing 4: `QuestionAnswerer.py` contains the `QuestionAnswerer` class which deals with the logic of answering parametric and counterfactual questions from a model

```
import logging

from transformers import AutoTokenizer, AutoModelForCausalLM,
    AutoModelForSeq2SeqLM, BatchEncoding
from torch import nn, tensor
from torch import FloatTensor, Tensor
import torch

# Dictionary of models, containing all of the models aliases and their respective
    models.
Model_dict = {
    'llama': 'meta-llama/Meta-Llama-3.1-8B-Instruct',
    'llama-70b': 'meta-llama/Meta-Llama-3.1-70B-Instruct',
    'llama-405b': 'meta-llama/Meta-Llama-3.1-405B-Instruct',
    'flan-t5': 'google/flan-t5-base',
    'flan-t5-small': 'google/flan-t5-small',
    'flan-t5-base': 'google/flan-t5-base',
    'flan-t5-large': 'google/flan-t5-large',
    'flan-t5-xl': 'google/flan-t5-xl',
    'flan-t5-xxl': 'google/flan-t5-xxl',
    'gemma': 'google/gemma-2-9b-it',
    'gemma-27b': 'google/gemma-2-27b-it',
    'falcon2': 'tiiuae/falcon-11b',
    'falcon-180b': 'tiiuae/falcon-180b-chat',
    'falcon-40b': 'tiiuae/falcon-40b-instruct',
    'falcon-7b': 'tiiuae/falcon-7b-instruct',
    'distilbert': 'distilbert/distilbert-base-uncased-distilled-squad',
    'roberta': 'FacebookAI/roberta-base',
    'roberta-large': 'FacebookAI/roberta-large',
    'roberta-squad': 'deepset/roberta-base-squad2',
    'mixtral': 'mistralai/Mixtral-8x22B-Instruct-v0.1',
    'dummy': '',
}

```

```python
# Virtual class containing a model.
# Derived classes should reimplement __init__ and logits.
class Model(nn.Module):
    name: str
    model_name: str
    device: str

    tokenizer: AutoTokenizer
    model: AutoModelForCausalLM

    # Construct a model from a certain name.
    # This should be the main constructor of models.
    @staticmethod
    def fromName(name: str, device: str = 'cpu') -> 'Model':
        if name == 'dummy':
            return DummyModel()

        if name in ('llama-70b', 'gemma-27b'):
            return LargeDecoderOnlyModel(name, device)

        if 't5' in name:
            return Seq2SeqModel(name, device)

        return DecoderOnlyModel(name, device)

    def __init__(self, name: str, device: str = 'cuda'):
        super().__init__()
        self.name = name
        self.model_name = Model_dict[name]
        self.device = device

    @torch.no_grad()
    def logits(self, query: BatchEncoding, answer: BatchEncoding) -> FloatTensor:
        raise NotImplementedError('logits called from generic Model class')

# Decoder-only model, such as llama.
class DecoderOnlyModel(Model):
    def __init__(self, name: str, device: str = 'cuda'):
        super().__init__(name, device)

        # self.prompt = 'Answer the following question in a few words and with no formatting.'
        # self.cf_prompt = 'Answer the following question using the previous context in a few words and with no formatting.'
        self.prompt = ''
        self.cf_prompt = ''

        kwargs = {}
        if 'llama' in name:
            kwargs = dict(
                pad_token = '<|reserved_special_token_0|>',
                padding_side = 'left',
            )
        elif 'gemma' in name:
            kwargs = dict(
                padding_side = 'right',
            )

        self.tokenizer = AutoTokenizer.from_pretrained(
            self.model_name,
            clean_up_tokenization_spaces = True,
            **kwargs,
```

```
        )

        logging.info(f'Loading model for {self.model_name} using {torch.cuda.device_count()} GPUs.')
        self.model = AutoModelForCausalLM.from_pretrained(
            self.model_name,
            device_map = 'auto' if self.device == 'cuda' else self.device,
            torch_dtype = torch.bfloat16,
            pad_token_id = self.tokenizer.pad_token_id,
            bos_token_id = self.tokenizer.bos_token_id,
            eos_token_id = self.tokenizer.eos_token_id,
            low_cpu_mem_usage = True,
        )
        self.model.eval()

    @torch.no_grad()
    def logits(self, query: BatchEncoding, answer: BatchEncoding) -> FloatTensor:
        w0 = query.input_ids.shape[1]
        w1 = answer.input_ids.shape[1]

        input_ids = torch.cat([query.input_ids, answer.input_ids], dim = 1)
        attention_mask = torch.cat([query.attention_mask, answer.attention_mask], dim = 1)

        return self.model(input_ids, attention_mask = attention_mask).logits[:, w0 - 1 : w0 + w1 - 1]

# Seq2Seq model, such as Flan-T5.
class Seq2SeqModel(Model):
    def __init__(self, name: str, device: str = 'cpu'):
        super().__init__(name, device)

        self.prompt = ''
        self.cf_prompt = ''

        kwargs = dict(
            padding_side = 'right',
        )
        self.tokenizer = AutoTokenizer.from_pretrained(
            self.model_name,
            clean_up_tokenization_spaces = True,
            **kwargs,
        )

        logging.info(f'Loading Seq2Seq model for {self.model_name} using {torch.cuda.device_count()} GPUs.')
        self.model = AutoModelForSeq2SeqLM.from_pretrained(
            self.model_name,
            device_map = 'auto' if self.device == 'cuda' else self.device,
            torch_dtype = torch.bfloat16,
            pad_token_id = self.tokenizer.pad_token_id,
            bos_token_id = self.tokenizer.bos_token_id,
            eos_token_id = self.tokenizer.eos_token_id,
            low_cpu_mem_usage = True,
        )
        self.model.eval()

    @staticmethod
    def pad(tensor: Tensor, length: int, value) -> Tensor:
        right = torch.full((tensor.shape[0], length - tensor.shape[1]), value)
        return torch.cat([tensor, right.to(tensor.device)], dim = 1)

```

```python
    @torch.no_grad()
    def logits(self, query: BatchEncoding, answer: BatchEncoding) -> FloatTensor:
        length = max(query.input_ids.shape[1], answer.input_ids.shape[1])

        input_ids = self.pad(query.input_ids, length, self.tokenizer.pad_token_id)
        attention_mask = self.pad(query.attention_mask, length, 0)
        decoder_input_ids = self.pad(self.model._shift_right(answer.input_ids), length, self.tokenizer.pad_token_id)

        return self.model(
            input_ids = input_ids,
            attention_mask = attention_mask,
            decoder_input_ids = decoder_input_ids,
        ).logits[:, : answer.input_ids.shape[1]]

# Large decoder-only model.
# Similar to DecoderOnlyModel, but eagerly deletes the model when the class is deleted.
# Assumes you need 2 GPUs to run this.
class LargeDecoderOnlyModel(DecoderOnlyModel):
    def __init__(self, name, device: str = 'cuda'):
        if torch.cuda.device_count() < 2:
            raise ValueError(f'At least two GPUs are needed to run {name}')

        super().__init__(name, device)

    def __del__(self):
        logging.info(f'Deleting large model {self.name}')
        del self.model
        torch.cuda.empty_cache()

# Dummy model, used for testing.
class DummyModel(Model):
    def __init__(self):
        nn.Module.__init__(self)
        self.name = 'dummy'
        self.tokenizer = self
        self.model = self
        self.sequences = ['dummy']
        self.logits = tensor([[[1., 2., 3.]]])

        self.bos_token_id = 0
        self.eos_token_id = 1
        self.pad_token_id = 2

    def to(self, *args, **kwargs):
        return self

    def __call__(self, *args, **kwargs):
        return self

    def generate(self, *args, **kwargs):
        return self

    def __getitem__(self, key):
        return self

    def decode(self, *args, **kwargs):
        return 'Dummy text'

    def batch_decode(self, *args, **kwargs):
        return ['Dummy Text 1', 'Dummy Text 2']
```

```python

    def shape(self):
        return (1, 2, 3)

# If called separately, just print the names of the models.
def main():
    print(f'{"Model Name":>15} | {"Huggingface Model":<40}')
    print((15 + 1) * '-' + '|' + (40 + 1) * '-')
    for name, model in Model_dict.items():
        print(f'{name:>15} | {model:<40}')

if __name__ == '__main__':
    main()
```

Listing 5: `Models.py` contains the list of models and includes code that differentiates them.

```python
import csv
import logging
import itertools
import random
import time
import typing

from collections import defaultdict
from dataclasses import dataclass
from typing import Optional

# Custom filter that does not print a log if it printed another one at most `rate_limit` seconds ago.
class LogTimeFilter(logging.Filter):
    def __init__(self):
        super().__init__()
        self.last_log = defaultdict(lambda: 0)

    def filter(self, record):
        if not hasattr(record, 'rate_limit'):
            return True

        current_time = time.time()
        if current_time - self.last_log[record.lineno] >= record.rate_limit:
            self.last_log[record.lineno] = current_time
            return True

        return False

# A question contains combines a base_question and an object into something that can be queried.
@dataclass
class Question:
    category: str
    obj: str
    base_question: str

    # Static constructor: return a question combining an object and an object if the category
    # matches; return None otherwise.
    @staticmethod
    def orNothing(obj: str, category: str, base_question: str) -> Optional['Question']:
        if not f'{{{category}}}' in base_question:
            return None
```

```

        return Question(obj = obj, category = category, base_question = base_question)

    # Return a query from the format of this Question.
    def format(self, *, prompt: Optional[str] = None, context: Optional[str] = None, use_question: bool = True, use_later: bool = True) -> str:
        [question, later] = self.base_question.format_map({self.category: self.obj}).split('?', 1)
        question += '?'

        formatted = ''
        if use_question:
            formatted = f'Q: {question.strip()}'

        if use_later:
            formatted = f'{formatted} A: {later.strip()}'

        if prompt is not None:
            formatted = f'{prompt} {formatted}'

        if context is not None:
            formatted = f'Context: [{later.strip()} {context}]. {formatted}'

        return formatted.strip()

# Returns the set product of a list of base question with the respective set of objects.
def combine_questions(base_questions: list[str], objects: list[dict[str, str]], lim_questions: Optional[int] = None) -> list[Question]:
    questions = []
    for bq in base_questions:
        for obj in objects:
            obj = Question.orNothing(obj = obj['object'], category = obj['category'], base_question = bq)
            if obj is None:
                continue

            questions.append(obj)

    if lim_questions is None:
        return questions

    keep_nums = {x: e for e, x in enumerate(random.sample(range(len(questions)), lim_questions))}
    short_questions = [questions[x] for x in keep_nums.keys()]

    return short_questions

# Given a list of questions and a list of answers, produce a list of integers that would provide the
# index to a randomly sampled counterfactual.
def sample_counterfactual_flips(questions: list[Question], answers: list[str]) -> list[int]:
    flips = [-1 for _ in questions]

    for q, es_iter in itertools.groupby(range(len(questions)), key = lambda e: questions[e].base_question):
        es = set(es_iter)

        for e in es:
            rest = [x for x in es if answers[x] != answers[e]]
```

```
            if not rest:
                logging.error(f'Unitary question "{q}". This means that all answers in this chunk are identical, and the results will be incorrect.')
                flips[e] = e
                continue

            flips[e] = random.choice(rest)
            assert answers[flips[e]] != answers[e]

    assert all(x != -1 for x in flips)
    return flips

# Chunk a list of question into batches of size or at most `max_batch_size`.
def chunk_questions(questions: list[Question], max_batch_size: int) -> list[list[Question]]:
    result: list[list[Question]] = []

    for q, chunk_iter in itertools.groupby(questions, key = lambda x: x.base_question):
        chunk = list(chunk_iter)
        if not result or len(chunk) + len(result[-1]) > max_batch_size:
            result.append([])

        result[-1].extend(chunk)

    return result

# Prints a CSV file with the questions and resulting answers.
def print_parametric_csv(out: typing.TextIO, questions: list[Question], answer: dict[str, str]):
    fieldnames = ['Num', 'Category', 'Base_Question', 'Thing', 'Question', 'Prefix'] + list(answer.keys())

    writer = csv.DictWriter(
        out,
        fieldnames = fieldnames,
        extrasaction = 'ignore',
        dialect = csv.unix_dialect,
        quoting = csv.QUOTE_MINIMAL,
    )
    writer.writeheader()

    for e, (question, *answers) in enumerate(itertools.zip_longest(questions, *answer.values())):
        question = typing.cast(Question, question)

        param = dict(zip(answer.keys(), answers))
        writer.writerow({'Num': str(e), 'Category': question.category, 'Base_Question': ''.join(question.base_question.partition('?')[0:2]), 'Thing': question.obj, 'Question': question.format(use_later = False), 'Prefix': question.format(use_question = False)} | param)
```

Listing 6: `Utils.py` contains various useful functions

```
import unittest
from unittest import TestCase
from unittest.mock import MagicMock

import torch
from torch import tensor

```

```python
from QuestionAnswerer import QuestionAnswerer

pad = 128002
class QuestionAnswererTests(unittest.TestCase):
    def setUp(self):
        self.qa = QuestionAnswerer('dummy', 'cpu', None)
        # self.qa.llm.tokenizer = MagicMock()
        self.qa.llm.tokenizer.pad_token_id = pad
        self.qa.llm.tokenizer.batch_decode = MagicMock(
            return_value = ['Hello how are you', 'Newline here', 'No stop string', '']
        )

    def test_winner(self):
        logits = tensor([
            [[0.0900, 0.2447, 0.6652], [0.6652, 0.2447, 0.0900], [0.2447, 0.6652, 0.0900]],
            [[0.2119, 0.2119, 0.5761], [0.2119, 0.2119, 0.5761], [0.2119, 0.2119, 0.5761]],
            [[0.6652, 0.2447, 0.0900], [0.2119, 0.5761, 0.2119], [0.5761, 0.2119, 0.2119]],
        ])

        expected_path = tensor([[2, 0, 1], [2, 2, 2], [0, 1, 0]])
        expected_probs = tensor([
            [0.6652, 0.6652, 0.6652],
            [0.5761, 0.5761, 0.5761],
            [0.6652, 0.5761, 0.5761],
        ])

        path, probs = self.qa.winner(logits)
        self.assertTrue(torch.equal(path, expected_path), msg = (path, expected_path))
        self.assertTrue(torch.allclose(probs, expected_probs), msg = (probs, expected_probs))

    def test_decode(self):
        path = tensor([
            [128000, 9906, 1268,  527,  499,    13,  358, 1097, 3815, 7060, 9901,  499,   13],
            [128000, 3648, 1074, 1618,  198, 54953,    0,   13, 1234, 1234, 1234, 1234, 1234],
            [128000, 2822, 3009,  925, 1234,  1234, 1234, 1234, 1234, 1234, 1234, 1234, 1234],
            [128000,   13, 1234, 1234, 1234,  1234, 1234, 1234, 1234, 1234, 1234, 1234, 1234],
        ])
        probs = tensor([
            [1., 3., 5., 7., 9., 11., 13., 15., 17., 19., 21., 23., 25.],
            [1., 3., 5., 7., 9., 11., 13., 15., 17., 19., 21., 23., 25.],
            [1., 3., 5., 7., 9., 11., 13., 15., 17., 19., 21., 23., 25.],
            [1., 3., 5., 7., 9., 11., 13., 15., 17., 19., 21., 23., 25.],
        ])

        expected_result = [
            'Hello how are you',
            'Newline here',
            'No stop string',
            '',
        ]
        expected_mean_probs = [5., 4., 13., 1.]

```

```
        result, mean_probs = self.qa.decode(path, probs)
        self.assertListEqual(expected_result, result)
        self.assertListEqual(expected_mean_probs, mean_probs)
```

Listing 7: `test_QuestionAnswerer.py` is used to test some of the complicated bits of logic in QuestionAnswerer.