



CITY
UNIVERSITY OF LONDON
— EST 1894 —

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	5
1.1	Problem Background	5
1.2	Thesis Questions & Objectives	6
1.2.1	Creating a representative dataset of questions	6
1.2.2	When does a model choose the provided context knowledge over its inherent knowledge?	7
1.2.3	Can we use the perplexity score of an answer to predict whether it came from inherent or contextual knowledge?	7
2	Context	8
2.1	Foundational Papers on Large Language Models	8
2.2	Papers working with RAG and contextual data	8
2.3	On disentangling parametric and context-augmented counterparametric knowledge	8
3	Methods	9
3.1	Creating a representative dataset of questions	9
3.2	When does a model choose the provided context knowledge over its inherent knowledge?	10
3.2.1	Model Selection	10
3.2.2	What type of answer does each model select for each question?	10
3.3	Can we use the perplexity score of an answer to predict whether it came from inherent or contextual knowledge?	12
3.3.1	Perplexity Score	12
3.3.2	Perplexity of the parametric answer with counterparametric context and vice-versa	13
3.3.3	Predicting whether an answer came from memory or from context	13
4	Results	15
4.1	Creating a representative dataset of questions	15
4.2	When does a model choose the provided context knowledge over its inherent knowledge?	16
4.3	Can we use the perplexity score of an answer to predict whether it came from inherent or contextual knowledge?	20
5	Discussion	23
5.1	Model type and memorised knowledge	23
5.2	Model size and memorised knowledge	23
5.3	Differences in perplexity scores for larger and smaller models	23
5.3.1	Can we use this to predict from where an answer came from?	23
5.4	Differences in distributions for different categories and questions.	23

6 Evaluations, Reflections, and Conclusions	24
6.1 Future Work	24
6.1.1 Knowledge Grounding in Retrieval-Augmented LMs	24
6.1.2 Further Memory Locator Prediction	24
6.1.3 Fine-tuning a LLM for a RAG Context	24
Glossary	25
Bibliography	26
Appendices	28
A Questions and objects used to form the queries	28
B Full Results for Each Question	35
C Grounder Usage and Documentation	35
D Source Code of the Experiments	36

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 Thesis Questions & Objectives

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

1.2.1 Creating a representative dataset of questions

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 models, we require a dataset with the following properties.

1. The dataset must contain questions that have short, unambiguous answers.
2. The questions must cover a large set of topics.
3. It must allow for the creation of counterparametric answers in the same format as correct ones to test contextual versus inherent knowledge.

The existing literature uses various existing question-and-answer datasets, none of which are useful for this research.*

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.

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.

*TODO: Maybe this entire subsection should go on Section 2 or Section 3.

1.2.2 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 answer picked depends on the type and size 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.

This thesis also gathers insights from answering this question on different categories and patterns of questions to find out if this depends on what is being asked.

1.2.3 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

As argued in Section 1.2.1, our codebase requires the creation of a new dataset of questions with three main properties.

1. The questions should have short and unambiguous answers.
2. They must cover a large set of topics, eras, and places.
3. They must allow for the creation of sensible counterparametric answers (different than what the model would normally answer) by having sets of questions with the same answer format.

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.

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.

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.

	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 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.

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.

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 3 contains an example of the shuffling done for this experiment while Table 4 contains an example of each of the three categories.

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

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 4: 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.

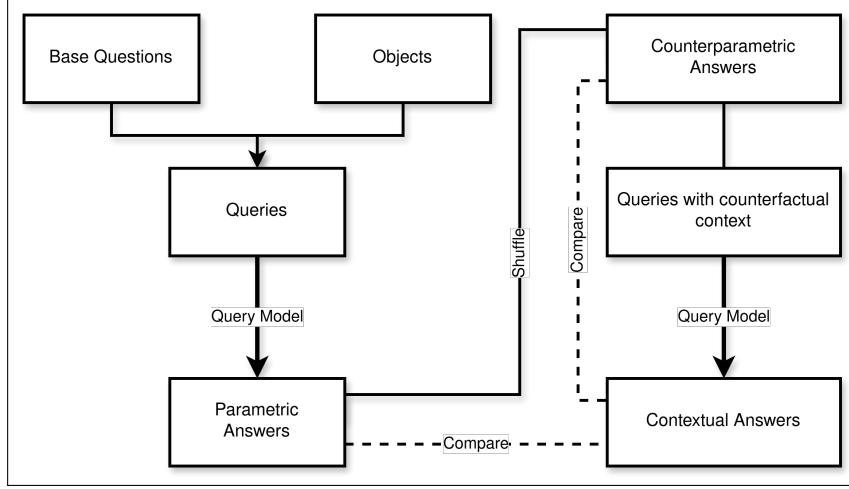


Figure 1: Example diagram of steps used to calculate the two sets of answers, *parametric* and *contextual*, and to compare them to answer the question in this objective. Many of the terms in this diagram are explained in the Glossary.

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, \dots, 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).

$$\text{NLL}(x_1, \dots, x_n | Q) = -\frac{1}{n} \sum_{i=1}^n \log_2 P(x_i | Q, x_1, \dots, x_{i-1}) \quad (1)$$

$$\text{PPL}(x_1, \dots, x_n | Q) = 2^{\text{NLL}(x_1, \dots, x_n | Q)} \quad (2)$$

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, \dots, 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, \dots, p_n and randomly sampled counterparametric answer q_1, \dots, 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.

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

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

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.

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, \dots, 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 $\text{PPL}(x_1, \dots, x_n \mid Q')$ 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.

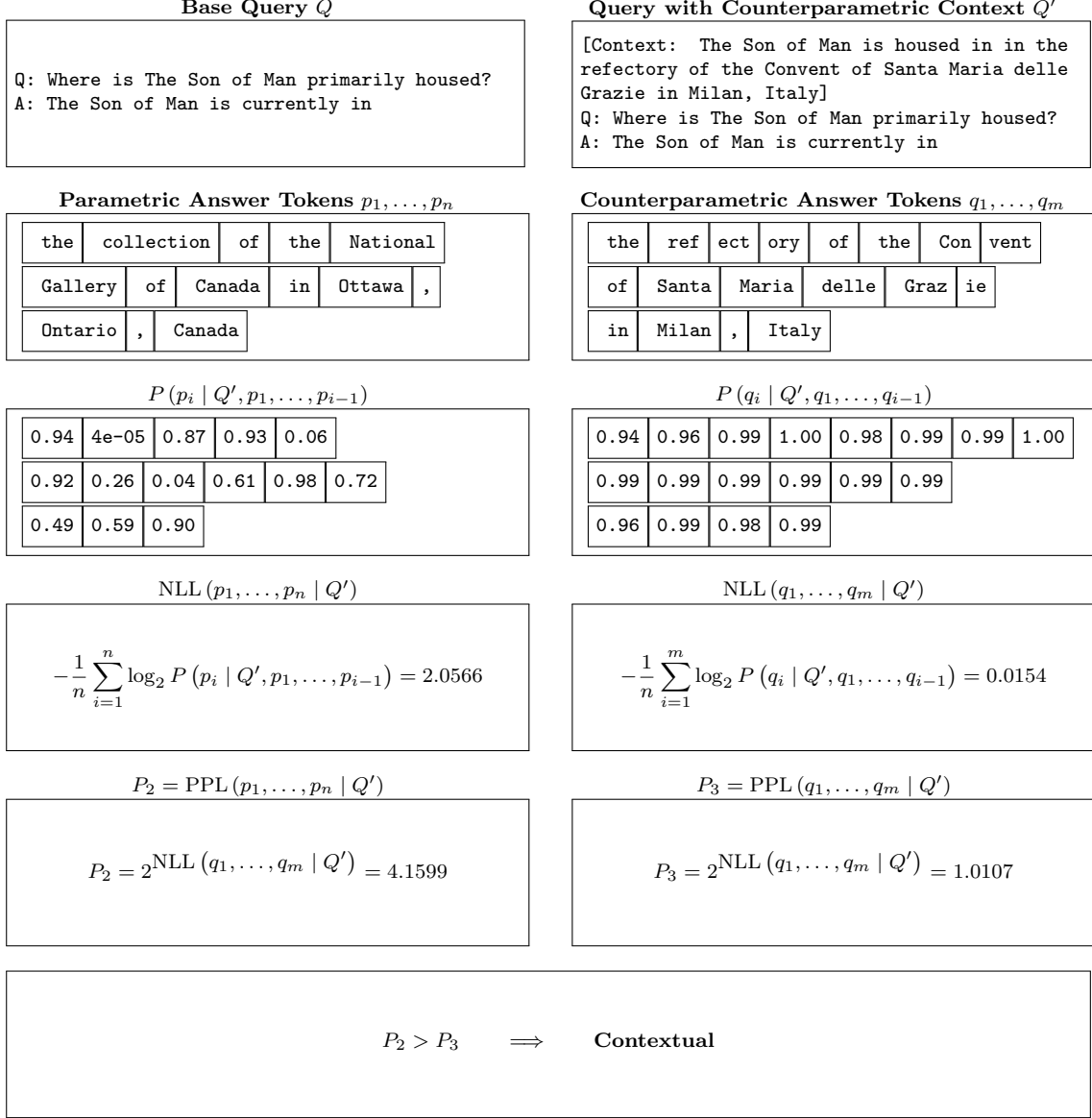


Figure 2: Example of perplexity calculation for the parametric and counterparametric answers in a query with the counterparametric context. Note that, due to teacher forcing, the calculation finds the probability of the next token p_i given the previous tokens of the searched answer p_1, \dots, p_{i-1} 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.

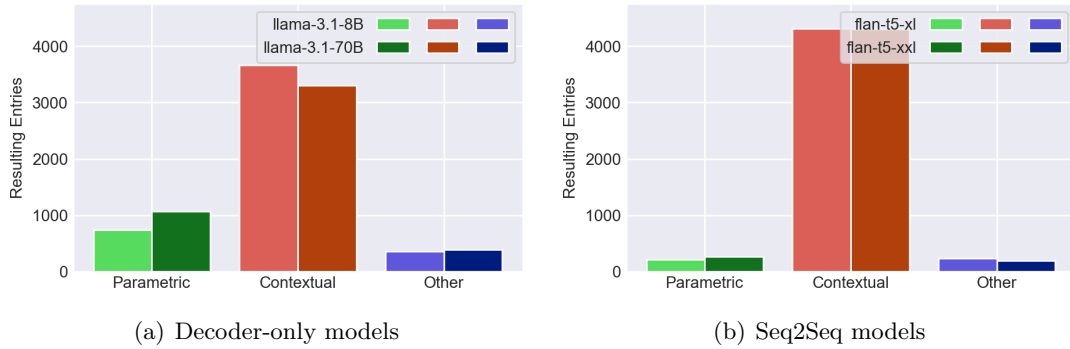


Figure 3: Results by type depending on which model; these are the same results as Table 7.

As hypothesised in Section 1.2.2, 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.

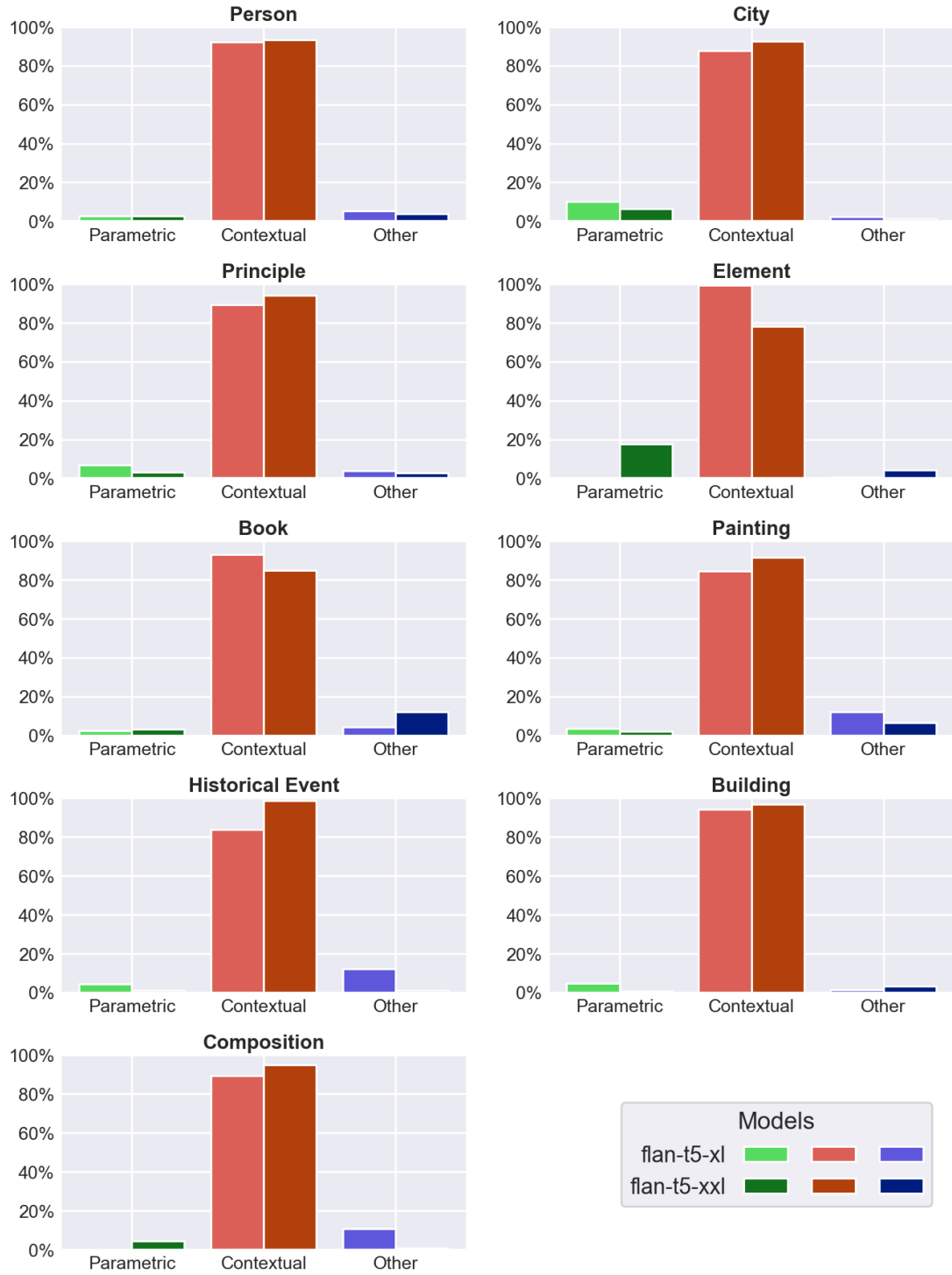


Figure 5: Results of running Seq2Seq models on the queried data, grouped by category. This plots the information shown in Table 9.

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

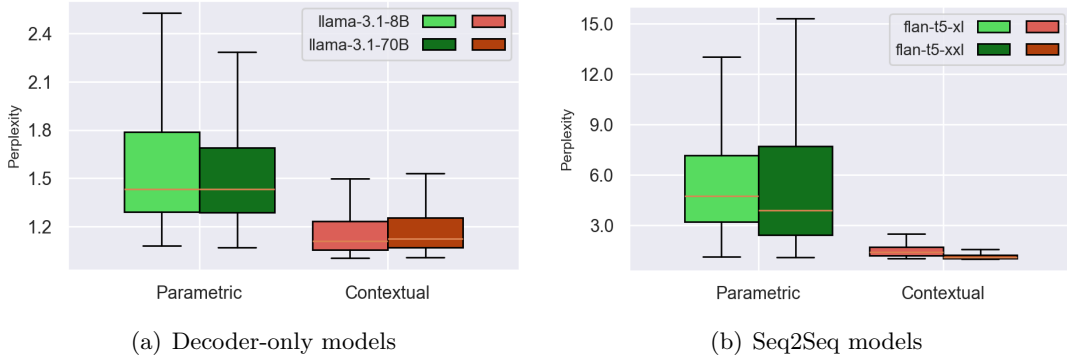


Figure 6: Perplexity distribution according to model type and size. These represent the same distributions as Tables 10 and 11.

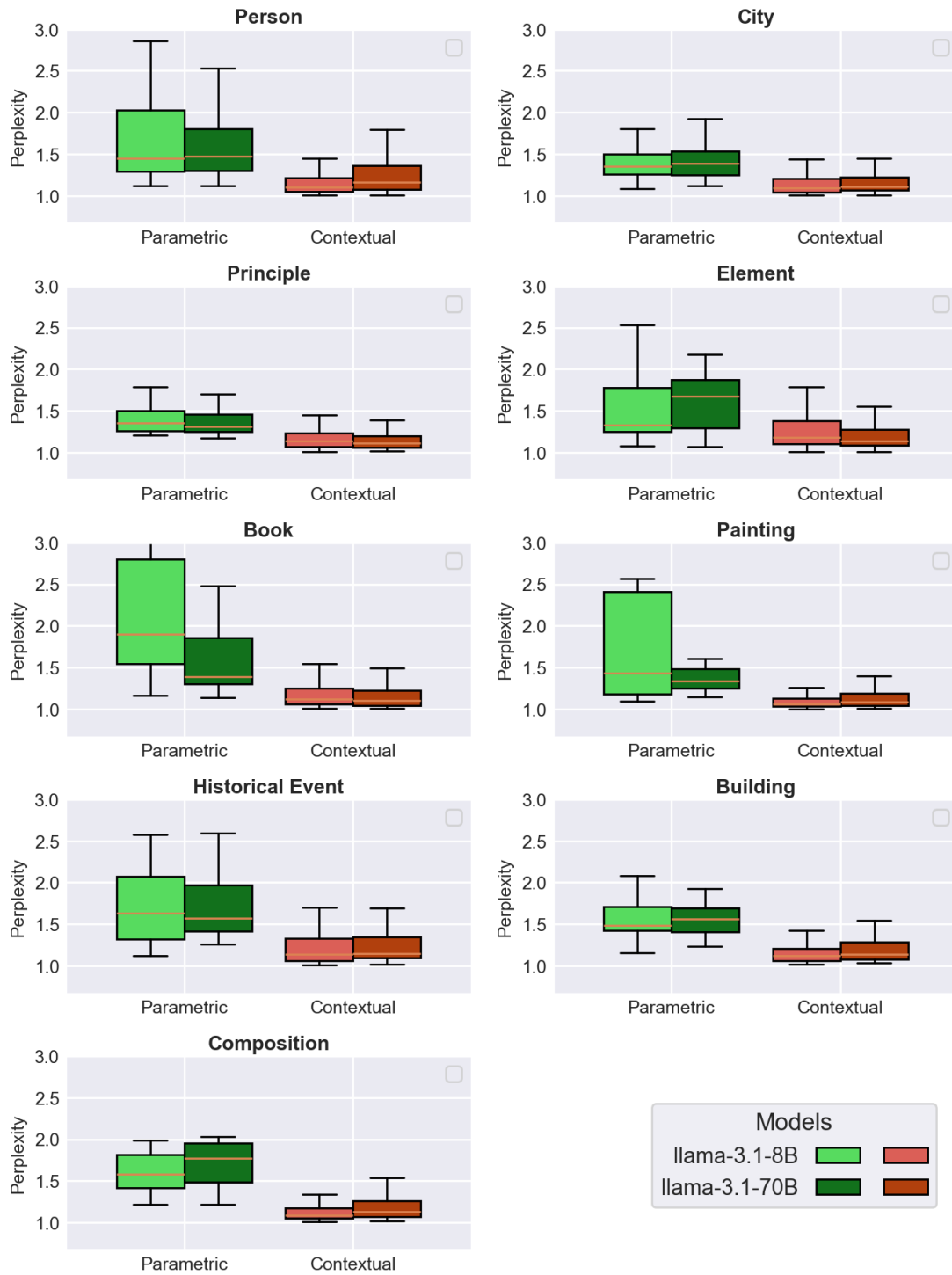


Figure 7: Box plots representing the distribution of the perplexities when running both Llama models, grouped by category.

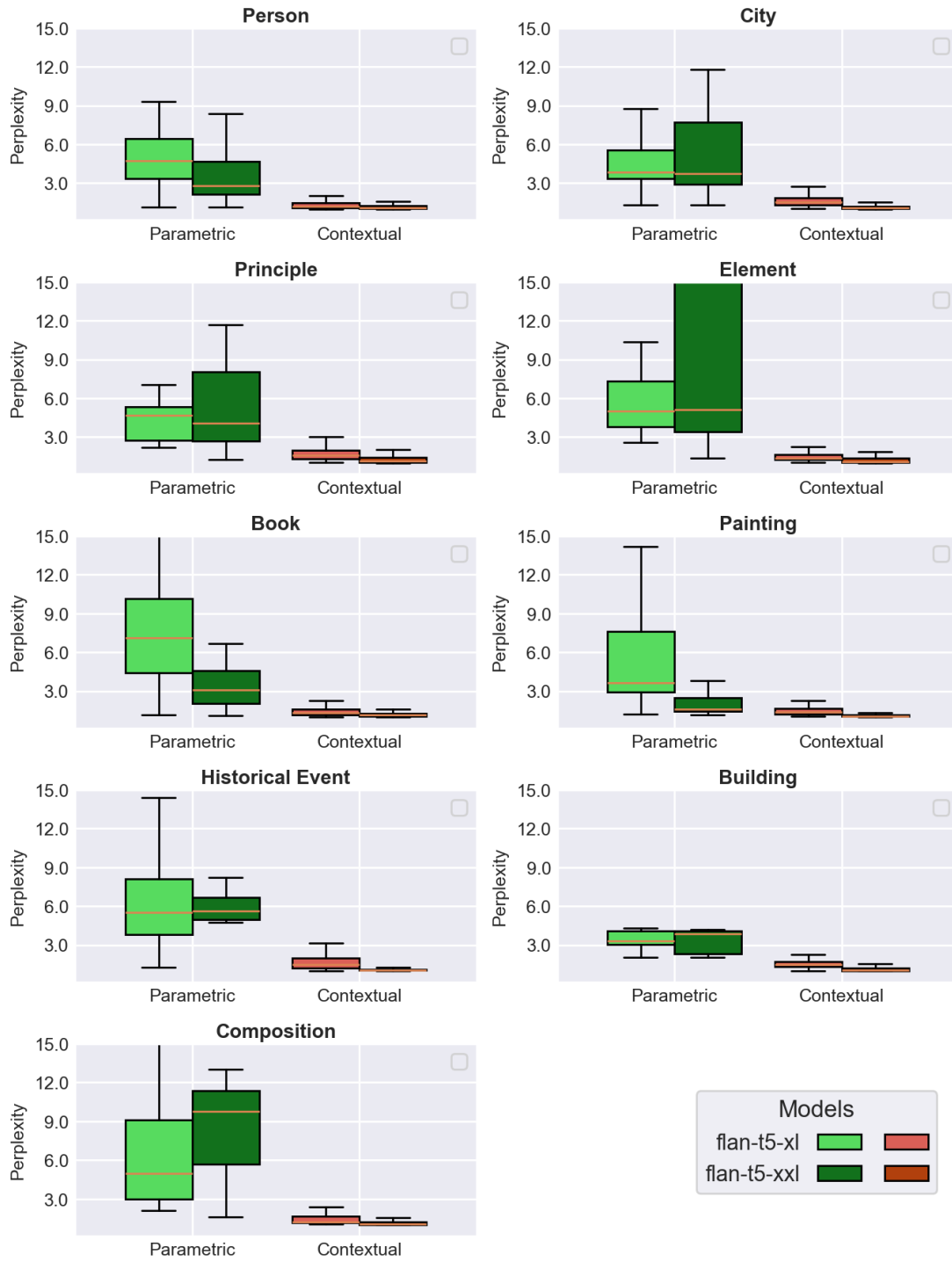


Figure 8: Box plots representing the distribution of the perplexities when running both Flan-T5 models, grouped by category.

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 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 (Izcard 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.

6.1.2 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.

6.1.3 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

- 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., Kuttler, H., Lewis, M., Yih, W.-t., Rocktaschel, 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
 Schrödinger 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
Potlatch 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
Für Elise,composition
Gymnopédies,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 combined 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[‡]>.

```
1 import warnings
2 warnings.simplefilter(action = 'ignore', category = FutureWarning)
3
4 from argparse import ArgumentParser
5 import csv
6 import logging
7 import random
8 import ipdb
9 import os
10 import sys
11 import wandb
12
13 from Models import Model_dict
14 from QuestionAnswerer import QuestionAnswerer
15 from Utils import print_parametric_csv, LogTimeFilter, combine_questions
16
17 def parse_args():
18     parser = ArgumentParser(
19         description = 'Combines questions and data and optionally provides
20         parametric data'
21     )
22
23     parser.add_argument('--debug', action = 'store_true', help = 'Go to IPDB
24     console on exception.')
25     parser.add_argument('--lim-questions', type = int, help = 'Question limit')
26     parser.add_argument('--device', choices = ['cpu', 'cuda'], default = 'cuda',
27     help = 'Inference device')
28     parser.add_argument('--models', type = str.lower, default = [], choices =
29     Model_dict.keys(), nargs = '+', metavar = 'model', help = 'Which model or
30     models to use for getting parametric data')
31     parser.add_argument('--offline', action = 'store_true', help = 'Tell HF to
32     run everything offline.')
33     parser.add_argument('--rand', action = 'store_true', help = 'Seed randomly')
34     parser.add_argument('--max-batch-size', type = int, default = 120, help =
35     'Maximum size of batches. All batches contain exactly the same question.')
36
37     parser.add_argument('--per-model', action = 'store_true', help = 'Write one
38     CSV per model in stdout.')
39     parser.add_argument('--output-dir', help = 'Return one CSV per model, and
40     save them to this directory.')
41
42     parser.add_argument('base_questions_file', type = open, help = 'File with
43     questions')
44     parser.add_argument('things_file', type = open, help = 'File with things to
45     combine')
46
47     args = parser.parse_args()
48
49     args.base_questions = [x.strip() for x in args.base_questions_file if any(not
50     y.isspace() for y in x)]
51     args.things = [{k: v for k, v in p.items()} for p in
52     csv.DictReader(args.things_file)]
53
54     del args.base_questions_file
```

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

```

42     del args.things_file
43
44     if args.per_model and args.output_dir:
45         raise ValueError('Only one of --per-model and --output-dir can be
specified.')
46
47     return args
48
49 def main(args):
50     logging.getLogger('transformers').setLevel(logging.ERROR)
51     logging.basicConfig(
52         format='[%asctime)s] %(message)s',
53         level=logging.INFO,
54         datefmt='%Y-%m-%d %H:%M:%S'
55     )
56     logging.getLogger().addFilter(LogTimeFilter())
57
58     if args.offline:
59         os.environ['TRANSFORMERS_OFFLINE'] = '1'
60     else:
61         wandb.init(project = 'knowledge-grounder', config = args)
62
63     logging.info('Getting questions')
64     questions = combine_questions(args.base_questions, args.things,
args.lim_questions)
65
66     if args.output_dir:
67         try:
68             os.mkdir(args.output_dir)
69         except FileExistsError:
70             pass
71
72     logging.info(f'About to answer {len(questions) * len(args.models) * 2}
questions in total.')
73     answers = {}
74     for model in args.models:
75         if not args.rand:
76             random.seed(0)
77
78         qa = QuestionAnswerer(model, device = args.device, max_length = 20,
max_batch_size = args.max_batch_size)
79         model_answers = qa.answerQueries(questions)
80         del qa
81
82         if args.output_dir:
83             empty = lambda s: sum([x == '' for x in model_answers[s]])
84             count = lambda s: sum([x == s for x in model_answers['comparison']])
85             logging.info(f"{model}:\t{empty('parametric')} empty parametrics,
{empty('counterfactual')} empty counterfactuals, {empty('contextual')} empty
contextuals")
86             logging.info(f"\t{count('Parametric')} parametrics,
{count('Contextual')} contextual, {count('Other')} others")
87
88             model_filename = os.path.join(args.output_dir, model + '.csv')
89             with open(model_filename, 'w') as out:
90                 print_parametric_csv(out, questions, model_answers)
91
92         elif args.per_model:
93             print_parametric_csv(sys.stdout, questions, model_answers)
94         else:
95             answers |= model_answers
96

```

```

97     if answers:
98         logging.info('Writing CSV')
99         print_parametric_csv(sys.stdout, questions, answers)
100
101 if __name__ == '__main__':
102     args = parse_args()
103     if not args.debug:
104         main(args)
105     else:
106         with ipdb.launch_ipdb_on_exception():
107             main(args)

```

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

```

1  import warnings
2  warnings.simplefilter(action = 'ignore', category = FutureWarning)
3
4  import logging
5  import math
6  import torch
7  import typing
8
9  from Models import Model
10 from typing import Optional, Union, Any
11 from Utils import Question, sample_counterfactual_flips, chunk_questions
12
13 from collections import defaultdict
14 from transformers import BatchEncoding
15
16 FloatTensor = torch.Tensor
17 LongTensor = torch.Tensor
18 BoolTensor = torch.Tensor
19
20 # A QuestionAnswerer is the main class to answer queries with a given model.
21 # Example Usage:
22 #   qa = QuestionAnswerer('llama', device = 'cuda', max_length = 20,
23 #   max_batch_size = 75)
24 #   output = qa.answerQueries(Utils.combine_questions(base_questions, objects))
25 # The list of models can be found in 'Model_dict' in 'Models.py'.
26 class QuestionAnswerer:
27     device: str
28     max_length: int
29     max_batch_size: int
30     llm: Model
31
32     def __init__(
33         self,
34         model: Union[str, Model],
35         device: str = 'cpu',
36         max_length: Optional[int] = None,
37         max_batch_size: Optional[int] = None,
38     ):
39         self.device = device
40         self.max_length = max_length or 100
41         self.max_batch_size = max_batch_size or 120
42
43         if type(model) == str:
44             model = Model.fromName(model, device = device)
45
46         model = typing.cast(Model, model)

```

```

46         self.llm = model
47
48         # Generated list of stop tokens: period, newline, and various different
end tokens.
49         stop_tokens = {'.', '\n'}
50         self.stop_token_ids = torch.tensor([
51             v
52             for k, v in self.llm.tokenizer.get_vocab().items()
53             if
54                 k in ['<start_of_turn>', '<end_of_turn>',
self.llm.tokenizer.special_tokens_map['eos_token']] or
55                 not stop_tokens.isdisjoint(self.llm.tokenizer.decode(v))
56         ]).to(self.device)
57
58         # Query data related to a list of questions, and return a dict with
information about these runs.
59         # Output elements:
60         # parametric: Parametric answer, as a string.
61         # base_proba: Perplexity of parametric answer in base query.
62         # counterfactual: Randomly selected counterfactual answer.
63         # base_cf_proba: Perplexity of counterfactual answer in base query.
64         # contextual: Contextual answer, as a string.
65         # ctx_proba: Perplexity of contextual answer.
66         # ctx_param_proba: Perplexity of parametric answer when running contextual
query.
67         # ctx_cf_proba: Perplexity of counterfactual answer when running contextual
query.
68         # comparison: Comparison between parametric and contextual answer. Where
does this answer come from?
69         # preference: Comparison between perplexity of parametric and counterfactual
answer on contextual query. Which one is the least surprising?
70         def answerChunk(self, questions: list[Question]) -> dict[str, Any]:
71             output: dict[str, Any] = {}
72
73             base_tokens = self.tokenise([q.format(prompt = self.llm.prompt) for q in
questions])
74             parametric = self.generate(base_tokens)
75
76             output['parametric'] = self.decode(parametric)
77             output['base_proba'] = self.perplexity(base_tokens, parametric)
78
79             flips = sample_counterfactual_flips(questions, output['parametric'])
80             counterfactual = parametric[flips]
81
82             output['counterfactual'] = self.decode(counterfactual)
83             output['base_cf_proba'] = self.perplexity(base_tokens, counterfactual)
84
85             output |= self.answerCounterfactuals(questions, output['counterfactual'],
parametric, counterfactual)
86
87             output['comparison'] = [
88                 'Parametric' if self.streq(a, p) else
89                 'Contextual' if self.streq(a, c) else
90                 'Other'
91                 for p, c, a in zip(output['parametric'], output['counterfactual'],
output['contextual'])
92             ]
93
94             output['preference'] = [
95                 'Parametric' if pp > cp else
96                 'Contextual'
97                 for pp, cp in zip(output['ctx_proba'], output['ctx_cf_proba'])

```



```

98     ]
99
100     return output
101
102
103     # Given a list of questions with assigned counterfactuals, run contextual
104     # queries and return
105     # a dictionary containing information about these runs.
106     # Parameter list:
107     #   questions: list of questions to ask.
108     #   counterfactuals: counterfactual answers, as string.
109     #   parametric: parametric answer, as set of tokens.
110     #   This will be used to calculate the perplexity of this answer with the
111     #   counterfactual context.
112     #   counterfactual: counterfactual answers, as a set of tokens.
113     #   This is necessary since the same string might have several encodings,
114     #   but we need exactly the same one generated by the model
115     #   in the first place.
116     def answerCounterfactuals(self, questions: list[Question], counterfactuals:
117     list[str], parametric: LongTensor, counterfactual: LongTensor) -> dict[str,
118     Any]:
119         output: dict[str, Any] = {}
120         ctx_tokens = self.tokenise([
121             q.format(prompt = self.llm.cf_prompt, context = context)
122             for q, context in zip(questions, counterfactuals)
123         ])
124         contextual = self.generate(ctx_tokens)
125
126         output['contextual'] = self.decode(contextual)
127         output['ctx_proba'] = self.perplexity(ctx_tokens, contextual)
128
129         output['ctx_param_proba'] = self.perplexity(ctx_tokens, parametric)
130         output['ctx_cf_proba'] = self.perplexity(ctx_tokens, counterfactual)
131
132         return output
133
134     # Answer a list of Questions: run the queries, gather counterfactual values,
135     # run the queries
136     # with counterfactual context, and return a 'dict' with information to print.
137     @torch.no_grad()
138     def answerQueries(self, questions: list[Question]) -> dict[str, Any]:
139         output: defaultdict[str, list[Any]] = defaultdict(lambda: [])
140
141         chunks = chunk_questions(questions, max_batch_size = self.max_batch_size)
142         logging.info(f'Answering {len(questions)} queries in {len(chunks)}
143         chunks.')
144
145         for e, chunk in enumerate(chunks, start = 1):
146             logging.info(f'Parsing chunk ({e} / {len(chunks)}), which has size
147             {len(chunk)}.', extra = {'rate_limit': 20})
148
149             chunk_output = self.answerChunk(chunk)
150
151             for k, v in chunk_output.items():
152                 output[k] += v
153
154         return dict(output)
155
156     # Tokenise a list of phrases.
157     # [n] -> (n, w)
158     def tokenise(self, phrases: list[str]) -> BatchEncoding:

```

```

152         return self.llm.tokenizer(
153             phrases,
154             return_tensors = 'pt',
155             return_attention_mask = True,
156             padding = True,
157         ).to(self.device)
158
159     # Generate an attention mask for a sequence of tokens.
160     # (n, w) -> (n, w)
161     def batch_encode(self, tokens: LongTensor) -> BatchEncoding:
162         attention_mask = tokens != self.llm.tokenizer.pad_token_id
163         return BatchEncoding(dict(
164             input_ids = tokens,
165             attention_mask = attention_mask,
166         ))
167
168     # Use Greedy decoding to generate an answer to a certain query.
169     # (n, w) -> (n, w)
170     def generate(self, query: BatchEncoding) -> LongTensor:
171         generated = self.llm.model.generate(
172             input_ids = query.input_ids,
173             attention_mask = query.attention_mask,
174             max_new_tokens = self.max_length,
175             min_new_tokens = self.max_length,
176             tokenizer = self.llm.tokenizer,
177             do_sample = False,
178             temperature = None,
179             top_p = None,
180             return_dict_in_generate = True,
181             pad_token_id = self.llm.tokenizer.pad_token_id,
182             eos_token_id = self.llm.tokenizer.eos_token_id,
183             bos_token_id = self.llm.tokenizer.bos_token_id,
184         )
185
186         # Ensure that all the sequences only contain <PAD> after their first stop
187         # token.
188         sequences = generated.sequences[:, -self.max_length:]
189         ignores = torch.cumsum(torch.isin(sequences, self.stop_token_ids), dim =
190         1) > 0
191         sequences[ignores] = self.llm.tokenizer.pad_token_id
192
193         return sequences
194
195     # Return the perplexity of a certain sequence of tokens being the answer to a
196     # certain query, as a list of floats in CPU.
197     # (n, w0), (n, w1) -> (n)
198     def perplexity(self, query: BatchEncoding, answer: LongTensor) -> list[float]:
199         probs = self.batch_perplexity(query, self.batch_encode(answer))
200         return probs.cpu().tolist()
201
202     # Return the perplexity of a certain sequence of tokens being the answer to a
203     # certain query.
204     # (n, w0), (n, w1) -> (n)
205     @torch.no_grad()
206     def batch_perplexity(self, query: BatchEncoding, answer: BatchEncoding) ->
207     FloatTensor:
208         entropies = self.llm.logits(query, answer).log_softmax(dim = 2)
209         entropies /= math.log(2)
210         probs = torch.where(
211             answer.input_ids == self.llm.tokenizer.pad_token_id,
212             torch.nan,
213             entropies.gather(index = answer.input_ids.unsqueeze(2), dim =

```

```

210         2).squeeze(2),
211     )
212
213     return torch.pow(2, -torch.nanmean(probs, dim = 1))
214
215     # Decode a sequence of tokens into a list of strings.
216     # (n, w) -> [n]
217     def decode(self, tokens: LongTensor) -> list[str]:
218         decoded = self.llm.tokenizer.batch_decode(
219             tokens,
220             skip_special_tokens = True,
221             clean_up_tokenization_spaces = True,
222         )
223         return [x.strip() for x in decoded]
224
225     # Compare strings for equality to later check whether an answer is parametric
226     # or contextual.
227     # For simplicity, we remove stop words and gather only the subset of words.
228     @staticmethod
229     def streq(a: str, b: str) -> bool:
230         a = a.lower().replace('the', '').replace(',', '').strip()
231         b = b.lower().replace('the', '').replace(',', '').strip()
232         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

```

1  import logging
2
3  from transformers import AutoTokenizer, AutoModelForCausalLM,
4      AutoModelForSeq2SeqLM, BatchEncoding
5  from torch import nn, tensor
6  from torch import FloatTensor, Tensor
7  import torch
8
9  # Dictionary of models, containing all of the models aliases and their respective
10  # models.
11  Model_dict = {
12      'llama': 'meta-llama/Meta-Llama-3.1-8B-Instruct',
13      'llama-70b': 'meta-llama/Meta-Llama-3.1-70B-Instruct',
14      'llama-405b': 'meta-llama/Meta-Llama-3.1-405B-Instruct',
15      'flan-t5': 'google/flan-t5-base',
16      'flan-t5-small': 'google/flan-t5-small',
17      'flan-t5-base': 'google/flan-t5-base',
18      'flan-t5-large': 'google/flan-t5-large',
19      'flan-t5-xl': 'google/flan-t5-xl',
20      'flan-t5-xxl': 'google/flan-t5-xxl',
21      'gemma': 'google/gemma-2-9b-it',
22      'gemma-27b': 'google/gemma-2-27b-it',
23      'falcon2': 'tiiuae/falcon-11b',
24      'falcon-180b': 'tiiuae/falcon-180b-chat',
25      'falcon-40b': 'tiiuae/falcon-40b-instruct',
26      'falcon-7b': 'tiiuae/falcon-7b-instruct',
27      'distilbert': 'distilbert/distilbert-base-uncased-distilled-squad',
28      'roberta': 'FacebookAI/roberta-base',
29      'roberta-large': 'FacebookAI/roberta-large',
30      'roberta-squad': 'deepset/roberta-base-squad2',
31      'mixtral': 'mistralai/Mixtral-8x22B-Instruct-v0.1',
32      'dummy': '',
33  }

```

```

33 # Virtual class containing a model.
34 # Derived classes should reimplement __init__ and logits.
35 class Model(nn.Module):
36     name: str
37     model_name: str
38     device: str
39
40     tokenizer: AutoTokenizer
41     model: AutoModelForCausalLM
42
43     # Construct a model from a certain name.
44     # This should be the main constructor of models.
45     @staticmethod
46     def fromName(name: str, device: str = 'cpu') -> 'Model':
47         if name == 'dummy':
48             return DummyModel()
49
50         if name in ('llama-70b', 'gemma-27b'):
51             return LargeDecoderOnlyModel(name, device)
52
53         if 't5' in name:
54             return Seq2SeqModel(name, device)
55
56         return DecoderOnlyModel(name, device)
57
58     def __init__(self, name: str, device: str = 'cuda'):
59         super().__init__()
60         self.name = name
61         self.model_name = Model_dict[name]
62         self.device = device
63
64     @torch.no_grad()
65     def logits(self, query: BatchEncoding, answer: BatchEncoding) -> FloatTensor:
66         raise NotImplementedError('logits called from generic Model class')
67
68 # Decoder-only model, such as llama.
69 class DecoderOnlyModel(Model):
70     def __init__(self, name: str, device: str = 'cuda'):
71         super().__init__(name, device)
72
73         # self.prompt = 'Answer the following question in a few words and with no
74         # self.cf_prompt = 'Answer the following question using the previous
75         # context in a few words and with no formatting.'
76         self.prompt = ''
77         self.cf_prompt = ''
78
79         kwargs = {}
80         if 'llama' in name:
81             kwargs = dict(
82                 pad_token = '<|reserved_special_token_0|>',
83                 padding_side = 'left',
84             )
85         elif 'gemma' in name:
86             kwargs = dict(
87                 padding_side = 'right',
88             )
89
90         self.tokenizer = AutoTokenizer.from_pretrained(
91             self.model_name,
92             clean_up_tokenization_spaces = True,
93             **kwargs,

```

```

93         )
94
95         logging.info(f'Loading model for {self.model_name} using
{torch.cuda.device_count()} GPUs.')
96         self.model = AutoModelForCausalLM.from_pretrained(
97             self.model_name,
98             device_map = 'auto' if self.device == 'cuda' else self.device,
99             torch_dtype = torch.bfloat16,
100             pad_token_id = self.tokenizer.pad_token_id,
101             bos_token_id = self.tokenizer.bos_token_id,
102             eos_token_id = self.tokenizer.eos_token_id,
103             low_cpu_mem_usage = True,
104         )
105         self.model.eval()
106
107     @torch.no_grad()
108     def logits(self, query: BatchEncoding, answer: BatchEncoding) -> FloatTensor:
109         w0 = query.input_ids.shape[1]
110         w1 = answer.input_ids.shape[1]
111
112         input_ids = torch.cat([query.input_ids, answer.input_ids], dim = 1)
113         attention_mask = torch.cat([query.attention_mask, answer.attention_mask],
dim = 1)
114
115         return self.model(input_ids, attention_mask = attention_mask).logits[:,
w0 - 1 : w0 + w1 - 1]
116
117 # Seq2Seq model, such as Flan-T5.
118 class Seq2SeqModel(Model):
119     def __init__(self, name: str, device: str = 'cpu'):
120         super().__init__(name, device)
121
122         self.prompt = ''
123         self.cf_prompt = ''
124
125         kwargs = dict(
126             padding_side = 'right',
127         )
128         self.tokenizer = AutoTokenizer.from_pretrained(
129             self.model_name,
130             clean_up_tokenization_spaces = True,
131             **kwargs,
132         )
133
134         logging.info(f'Loading Seq2Seq model for {self.model_name} using
{torch.cuda.device_count()} GPUs.')
135         self.model = AutoModelForSeq2SeqLM.from_pretrained(
136             self.model_name,
137             device_map = 'auto' if self.device == 'cuda' else self.device,
138             torch_dtype = torch.bfloat16,
139             pad_token_id = self.tokenizer.pad_token_id,
140             bos_token_id = self.tokenizer.bos_token_id,
141             eos_token_id = self.tokenizer.eos_token_id,
142             low_cpu_mem_usage = True,
143         )
144         self.model.eval()
145
146     @staticmethod
147     def pad(tensor: Tensor, length: int, value) -> Tensor:
148         right = torch.full((tensor.shape[0], length - tensor.shape[1]), value)
149         return torch.cat([tensor, right.to(tensor.device)], dim = 1)
150

```

```

151     @torch.no_grad()
152     def logits(self, query: BatchEncoding, answer: BatchEncoding) -> FloatTensor:
153         length = max(query.input_ids.shape[1], answer.input_ids.shape[1])
154
155         input_ids = self.pad(query.input_ids, length, self.tokenizer.pad_token_id)
156         attention_mask = self.pad(query.attention_mask, length, 0)
157         decoder_input_ids = self.pad(self.model._shift_right(answer.input_ids),
158                                     length, self.tokenizer.pad_token_id)
159
160         return self.model(
161             input_ids = input_ids,
162             attention_mask = attention_mask,
163             decoder_input_ids = decoder_input_ids,
164         ).logits[:, : answer.input_ids.shape[1]]
165
166 # Large decoder-only model.
167 # Similar to DecoderOnlyModel, but eagerly deletes the model when the class is
168 # deleted.
169 # Assumes you need 2 GPUs to run this.
170 class LargeDecoderOnlyModel(DecoderOnlyModel):
171     def __init__(self, name, device: str = 'cuda'):
172         if torch.cuda.device_count() < 2:
173             raise ValueError(f'At least two GPUs are needed to run {name}')
174
175         super().__init__(name, device)
176
177     def __del__(self):
178         logging.info(f'Deleting large model {self.name}')
179         del self.model
180         torch.cuda.empty_cache()
181
182 # Dummy model, used for testing.
183 class DummyModel(Model):
184     def __init__(self):
185         nn.Module.__init__(self)
186         self.name = 'dummy'
187         self.tokenizer = self
188         self.model = self
189         self.sequences = ['dummy']
190         self.logits = tensor([[[1., 2., 3.]]])
191
192         self.bos_token_id = 0
193         self.eos_token_id = 1
194         self.pad_token_id = 2
195
196     def to(self, *args, **kwargs):
197         return self
198
199     def __call__(self, *args, **kwargs):
200         return self
201
202     def generate(self, *args, **kwargs):
203         return self
204
205     def __getitem__(self, key):
206         return self
207
208     def decode(self, *args, **kwargs):
209         return 'Dummy text'
210
211     def batch_decode(self, *args, **kwargs):
212         return ['Dummy Text 1', 'Dummy Text 2']

```

```

211
212     def shape(self):
213         return (1, 2, 3)
214
215 # If called separately, just print the names of the models.
216 def main():
217     print(f'{"Model Name":>15} | {"Huggingface Model":<40}')
218     print((15 + 1) * '-' + '|' + (40 + 1) * '-')
219     for name, model in Model_dict.items():
220         print(f'{name:>15} | {model:<40}')
221
222 if __name__ == '__main__':
223     main()

```

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

```

1  import csv
2  import logging
3  import itertools
4  import random
5  import time
6  import typing
7
8  from collections import defaultdict
9  from dataclasses import dataclass
10 from typing import Optional
11
12 # Custom filter that does not print a log if it printed another one at most
13   'rate_limit' seconds ago.
14 class LogTimeFilter(logging.Filter):
15     def __init__(self):
16         super().__init__()
17         self.last_log = defaultdict(lambda: 0)
18
19     def filter(self, record):
20         if not hasattr(record, 'rate_limit'):
21             return True
22
23         current_time = time.time()
24         if current_time - self.last_log[record.lineno] >= record.rate_limit:
25             self.last_log[record.lineno] = current_time
26             return True
27
28         return False
29
30 # A question contains combines a base_question and an object into something that
31   can be queried.
32 @dataclass
33 class Question:
34     category: str
35     obj: str
36     base_question: str
37
38     # Static constructor: return a question combining an object and an object if
39     the category
40     # matches; return None otherwise.
41     @staticmethod
42     def orNothing(obj: str, category: str, base_question: str) ->
43     Optional['Question']:
44         if not f'{{{category}}}' in base_question:
45             return None

```

```

42
43     return Question(obj = obj, category = category, base_question =
base_question)
44
45     # Return a query from the format of this Question.
46     def format(self, *, prompt: Optional[str] = None, context: Optional[str] =
None, use_question: bool = True, use_later: bool = True) -> str:
47         [question, later] = self.base_question.format_map({self.category:
self.obj}).split('?', 1)
48         question += '??'
49
50         formatted = ''
51         if use_question:
52             formatted = f'Q: {question.strip()}'
53
54         if use_later:
55             formatted = f'{formatted} A: {later.strip()}'
56
57         if prompt is not None:
58             formatted = f'{prompt} {formatted}'
59
60         if context is not None:
61             formatted = f'Context: [{later.strip()} {context}]. {formatted}'
62
63         return formatted.strip()
64
65     # Returns the set product of a list of base question with the respective set of
objects.
66     def combine_questions(base_questions: list[str], objects: list[dict[str, str]],
lim_questions: Optional[int] = None) -> list[Question]:
67         questions = []
68         for bq in base_questions:
69             for obj in objects:
70                 obj = Question.orNothing(obj = obj['object'], category =
obj['category'], base_question = bq)
71                 if obj is None:
72                     continue
73
74                 questions.append(obj)
75
76         if lim_questions is None:
77             return questions
78
79         keep_nums = {x: e for e, x in enumerate(random.sample(range(len(questions)),
lim_questions))}
80         short_questions = [questions[x] for x in keep_nums.keys()]
81
82         return short_questions
83
84     # Given a list of questions and a list of answers, produce a list of integers
that would provide the
85     # index to a randomly sampled counterfactual.
86     def sample_counterfactual_flips(questions: list[Question], answers: list[str]) ->
list[int]:
87         flips = [-1 for _ in questions]
88
89         for q, es_iter in itertools.groupby(range(len(questions)), key = lambda e:
questions[e].base_question):
90             es = set(es_iter)
91
92             for e in es:
93                 rest = [x for x in es if answers[x] != answers[e]]

```



```

94         if not rest:
95             logging.error(f'Unitary question "{q}". This means that all
106             answers in this chunk are identical, and the results will be incorrect.')
107             flips[e] = e
108             continue
109
110             flips[e] = random.choice(rest)
111             assert answers[flips[e]] != answers[e]
112
113         assert all(x != -1 for x in flips)
114         return flips
115
116 # Chunk a list of question into batches of size or at most 'max_batch_size'.
117 def chunk_questions(questions: list[Question], max_batch_size: int) ->
118     list[list[Question]]:
119     result: list[list[Question]] = []
120
121     for q, chunk_iter in itertools.groupby(questions, key = lambda x:
122         x.base_question):
123         chunk = list(chunk_iter)
124         if not result or len(chunk) + len(result[-1]) > max_batch_size:
125             result.append([])
126
127             result[-1].extend(chunk)
128
129     return result
130
131 # Prints a CSV file with the questions and resulting answers.
132 def print_parametric_csv(out: typing.TextIO, questions: list[Question], answer:
133     dict[str, str]):
134     fieldnames = ['Num', 'Category', 'Base_Question', 'Thing', 'Question',
135         'Prefix'] + list(answer.keys())
136
137     writer = csv.DictWriter(
138         out,
139         fieldnames = fieldnames,
140         extrasaction = 'ignore',
141         dialect = csv.unix_dialect,
142         quoting = csv.QUOTE_MINIMAL,
143     )
144     writer.writeheader()
145
146     for e, (question, *answers) in enumerate(itertools.zip_longest(questions,
147         *answer.values())):
148         question = typing.cast(Question, question)
149
150         param = dict(zip(answer.keys(), answers))
151         writer.writerow({'Num': str(e), 'Category': question.category,
152             'Base_Question': ''.join(question.base_question.partition('?')[0:2]),
153             'Thing': question.obj, 'Question': question.format(use_later = False),
154             'Prefix': question.format(use_question = False)} | param)

```

Listing 6: Utils.py contains various useful functions

```

1 import unittest
2 from unittest import TestCase
3 from unittest.mock import MagicMock
4
5 import torch
6 from torch import tensor
7

```

```

8 from QuestionAnswerer import QuestionAnswerer
9
10 pad = 128002
11 class QuestionAnswererTests(unittest.TestCase):
12     def setUp(self):
13         self.qa = QuestionAnswerer('dummy', 'cpu', None)
14         # self.qa.llm.tokenizer = MagicMock()
15         self.qa.llm.tokenizer.pad_token_id = pad
16         self.qa.llm.tokenizer.batch_decode = MagicMock(
17             return_value = ['Hello how are you', 'Newline here', 'No stop
string', ''])
18     )
19
20     def test_winner(self):
21         logits = tensor([
22             [[0.0900, 0.2447, 0.6652], [0.6652, 0.2447, 0.0900], [0.2447, 0.6652,
0.0900]],
23             [[0.2119, 0.2119, 0.5761], [0.2119, 0.2119, 0.5761], [0.2119, 0.2119,
0.5761]],
24             [[0.6652, 0.2447, 0.0900], [0.2119, 0.5761, 0.2119], [0.5761, 0.2119,
0.2119]],
25         ])
26
27         expected_path = tensor([[2, 0, 1], [2, 2, 2], [0, 1, 0]])
28         expected_probs = tensor([
29             [0.6652, 0.6652, 0.6652],
30             [0.5761, 0.5761, 0.5761],
31             [0.6652, 0.5761, 0.5761],
32         ])
33
34         path, probs = self.qa.winner(logits)
35         self.assertTrue(torch.equal(path, expected_path), msg = (path,
expected_path))
36         self.assertTrue(torch.allclose(probs, expected_probs), msg = (probs,
expected_probs))
37
38     def test_decode(self):
39         path = tensor([
40             [128000, 9906, 1268, 527, 499, 13, 358, 1097, 3815, 7060, 9901,
499, 13],
41             [128000, 3648, 1074, 1618, 198, 54953, 0, 13, 1234, 1234, 1234,
1234, 1234],
42             [128000, 2822, 3009, 925, 1234, 1234, 1234, 1234, 1234, 1234, 1234,
1234, 1234],
43             [128000, 13, 1234, 1234, 1234, 1234, 1234, 1234, 1234, 1234, 1234,
1234, 1234],
44         ])
45         probs = tensor([
46             [1., 3., 5., 7., 9., 11., 13., 15., 17., 19., 21., 23., 25.],
47             [1., 3., 5., 7., 9., 11., 13., 15., 17., 19., 21., 23., 25.],
48             [1., 3., 5., 7., 9., 11., 13., 15., 17., 19., 21., 23., 25.],
49             [1., 3., 5., 7., 9., 11., 13., 15., 17., 19., 21., 23., 25.],
50         ])
51
52         expected_result = [
53             'Hello how are you',
54             'Newline here',
55             'No stop string',
56             '',
57         ]
58         expected_mean_probs = [5., 4., 13., 1.]
59

```

```
60         result, mean_probs = self.qa.decode(path, probs)
61         self.assertEqual(expected_result, result)
62         self.assertEqual(expected_mean_probs, mean_probs)
```

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