



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

In recent years large language models have exploded in quality and prevalence, and they have become crucial for work in a wide range of areas. However, their tendency to produce hallucinations presents a critical challenge in context where accuracy is crucial.

Retrieval-Augmented Generation (RAG), which leverages external information to provide more accurate and contextually appropriate responses, has been proposed as a solution to this issue. However, this solution is far from perfect as it's unclear when a large language model will choose the information provided by the RAG model over its learned, parametric knowledge.

This thesis explores the *Knowledge Grounding* of various large language models. In particular, it attempts to answer a simple research question: **when would a model choose an answer using its parametric or contextual knowledge, and why?**

To investigate this, we developed a diverse dataset comprising questions from various topics and globally representative data, which we used to construct queries with counterparametric context across four models of different types and sizes. We later analyse the perplexity of these answers to understand the level of surprise that a model has when presented with the parametric or contextual answer.

Our findings suggest that, when using RAG-enhanced systems, smaller and simpler models might produce fewer hallucinations. In particular, the smaller models **Meta-Llama-3.1-8B** and **Flan-T5-XL** tend to have better knowledge grounding and fewer hallucinations than their larger versions. Additionally, we propose novel methods for determining whether a given response originates from the RAG context or the model's internal memory using its resulting perplexity, which may indicate a potential hallucination.

This thesis forms the foundational part of a broader project aimed at publishing a comprehensive study on knowledge grounding in retrieval-augmented language models, as outlined in the preprint "Knowledge Grounding in Retrieval-Augmented LMs: An Empirical Study" (Whitehouse et al. 2023). We build on existing literature, incorporating the use of counterparametric context in queries, to advance our understanding of this phenomenon."

Contents

1	Introduction and Objectives	5
1.1	Problem Background	5
1.2	Research Question	5
1.3	Research Objectives	6
1.3.1	Creating a representative dataset of questions	6
1.3.2	When does a model choose the provided context knowledge over its inherent knowledge?	6
1.3.3	Can we use the perplexity score of an answer to predict whether it came from inherent or contextual knowledge?	6
2	Context	7
2.1	Foundational Papers on Large Language Models	7
2.2	Retrieval-Augmented Generation	8
2.3	Knowledge Grounding on Queries with Added Context	8
3	Methods	10
3.1	Creating a representative dataset of questions	10
3.1.1	Dataset Description	10
3.1.2	Dataset Creation	11
3.2	Understanding when a model chooses contextual knowledge versus parametric knowledge	12
3.2.1	Model Selection	12
3.2.2	What type of answer does each model select for each question?	13
3.3	Studying the Role of Perplexity Scores of the Provided Answers	16
3.3.1	Perplexity Score	16
3.3.2	Perplexity of the parametric answer with counterparametric context and vice-versa	16
3.3.3	Predicting whether an answer came from memory or from context	17
4	Results	19
4.1	Creating a representative dataset of questions	19
4.2	When does a model choose the provided context knowledge over its inherent knowledge?	20
4.3	Can we use the perplexity score of an answer to predict whether it came from inherent or contextual knowledge?	24
5	Discussion	27
5.1	Model type and memorised knowledge	27
5.2	Model size and memorised knowledge	27
5.3	Differences in perplexity scores for larger and smaller models	27
5.3.1	Can we use this to predict from where an answer came from?	27
5.4	Differences in distributions for different categories and questions.	27

6	Evaluations, Reflections, and Conclusions	28
6.1	Future Work	28
6.1.1	Better Categorisation of the Answers	28
6.1.2	Knowledge Grounding in Retrieval-Augmented LMs	28
6.1.3	Further Memory Locator Prediction	29
6.1.4	Fine-tuning a LLM for a RAG Context	29
	Glossary	30
	Bibliography	31
	Appendices	34
A	Questions and objects used to form the queries	34
B	Full Results for Each Question	41
C	Grounder Usage and Documentation	41
D	Source Code of the Experiments	42

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, ?), 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 summarizes key articles that informed this research.

2.1 Foundational Papers on Large Language Models

Large language models have exploded in popularity since the development of the transformer architecture (Vaswani et al. 2017). This architecture relies entirely on self-attention mechanisms rather than recurrent or convolutional layers, which allow the model to weigh the importance of different words in a sequence relative to each other, irrespective of their position. This mechanism enables the model to capture complex dependencies and relationships across long sequences more effectively than traditional models.

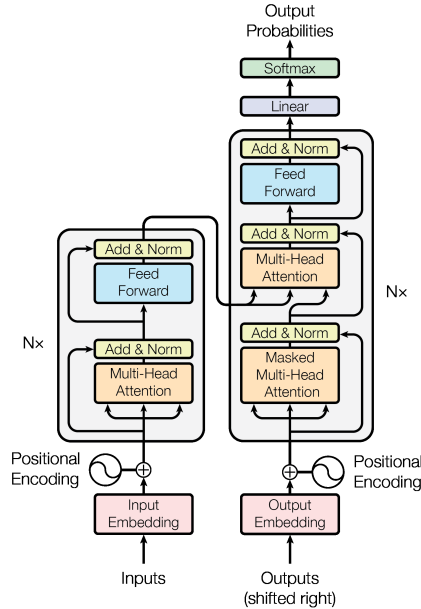


Figure 1: Transformer Architecture, from “Attention is all you need” (Vaswani et al. 2017).

GPT models (Radford & Narasimhan 2018) improve upon this architecture by running a supervised task-specific fine-tuning round after the unsupervised pre-training on large amount of test data. Later models, starting from GPT-2, use zero-shot transfer learning to improve their performance (Radford et al. 2019). Zero-shot learning (Norouzi et al. 2014) trains a model to perform a task without having been explicitly trained on examples of that task; instead, it leverages knowledge gained from pre-training to infer and generalise to new tasks with the context on the prompt.

By adding only a few examples of the task at hand, a model can use few-shot learning to

improve generalisation to new tasks with a limited amount of labelled data. GPT-3 uses this to understand the structure and nature of a task with a few examples (Brown et al. 2020)

TODO: Talk about T5, Flan-T5, and Llama here.

Despite these improvements, the models themselves are still uncalibrated and not very good at question answering (Jiang et al. 2021). Hallucinations are common, even for short Q&A tasks

2.2 Retrieval-Augmented Generation

Large pre-trained language models store factual knowledge in their parameters. Their ability to access this knowledge is limited, which affects their performance on knowledge-intensive tasks.

Retrieval-Augmented Generation (RAG) attempts to solve this problem by adding extra non-parametric context gathered from an index with a retriever that’s trained independently (Lewis et al. 2020). This retrieved context is fed back to the original query by adding context, which generates a combined representation which contains both the query and the original context. An overview of this method is presented in Figure 2.

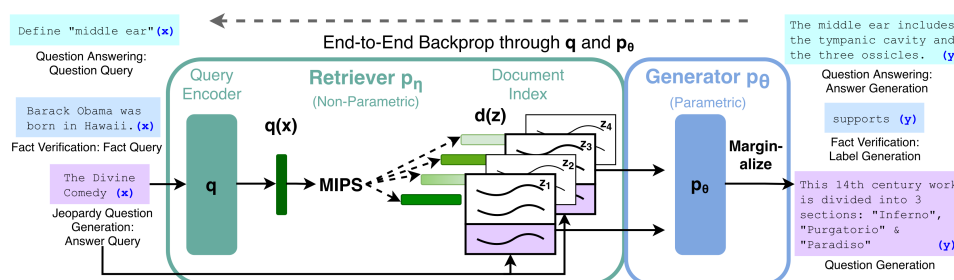


Figure 2: An overview of the RAG approach, combining a pre-trained retriever with a pre-trained model. From “Retrieval-Augmented Generation for Knowledge-Intensive NLP tasks” (Lewis et al. 2020)

RAG can be effective in preventing hallucinations by incorporating relevant external information into the generation process, ensuring that responses are more grounded in factual data from a knowledge base. However, this is not perfect and an even after adding a correct context the model could answer data from its internal memory.

2.3 Knowledge Grounding on Queries with Added Context

Knowledge grounding is the process of ensuring that information generated by a language model is consistent with a reliable external source of knowledge. In the context of RAG-enhanced queries, this can include ensuring that answers are consistent with the knowledge in the index rather than in the model’s inherent knowledge.

Various attempts have been made at understanding how good the knowledge grounding of a RAG system is, and on what is the optimal configuration of RAG.

In “RAGGED: Towards Informed Design of Retrieval Augmented Systems” (Hsia et al. 2024), the authors create a framework to evaluate these systems and find that different models suit substantially varied RAG setups. In particular, the performance of the models decreases strongly in Decoder-only models such as Llama when the context passages provides more context, while this does not happen on Seq2Seq models such as Flan.

“Characterizing Mechanisms for Factual Recall in Language Models” (Yu et al. 2023), which is one of the main sources for this thesis, finds that when there is disagreement between the model’s knowledge and the provided context then the type and size of the large language model will greatly affect the probability of the model choosing to use the context (which is much less prone to hallucinating) as an answer.

This paper also introduces a novel way to test this hypothesis by creating a dataset with questions and counterfactual context, an example of which is shown in Listing 1.

The capital of {country} is {in-context city}
Q: What is the capital of {country}?
A:

Listing 1: Example of queries used in (Yu et al. 2023). These queries form the basis and inspiration for the dataset creation done in this thesis

By adding counterparametric information to the context, this method allows us to understand whether an answer came is parametric (that is, came from the memory of the model) or contextual (that is, came from the provided context).

3 Methods

3.1 Creating a representative dataset of questions

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.

3.1.1 Dataset Description

The dataset we aim to create for this research is designed to be a comprehensive and versatile tool for evaluating large language models. By choosing this criteria when selecting questions, we ensure that the dataset will provide meaningful insights into knowledge grounding of large language models across a wide spectrum of knowledge domains.

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 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 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. We believe that none of these datasets are a good fit for this research for not following some of the three desired properties. However, understanding them can be useful when designing 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.

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.

3.2 Understanding when a model chooses contextual knowledge versus parametric 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 3.1 into four models of different types and sizes, which are listed in Table 2.

	Seq2Seq Model	Decoder-Only Model
Smaller	Flan-T5-XL	Meta-Llama-3.1-8B-Instruct
Larger	Flan-T5-XXL	Meta-Llama-3.1-70B-Instruct

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

Both Sequence-to-Sequence models are based on T5 models (Raffel et al. 2020), which employ an encoder-decoder architecture: while an encoder processed the input sequence into a context vector, and an decoder generates an input sequence from this vector. The Flan-T5 models are fine-tuned to follow instructions, and have improved zero-shot performance compared to the original T5 models (Chung et al. 2022).

Flan-T5-XL contains approximately 3 billion parameters, and is considerably bigger than the base Flan-T5 model, which should improve the accuracy of its parametric answers.

Flan-T5-XXL contains 11 billion parameters, so it should have higher accuracy on the parametric answers as the XL model. However, how the higher amount of parameters

will affect the experiments that we are running is still unknown.

Unlike Seq2Seq models, Decoder-only models focus on the generation of text based on the previous tokens. In a sense, given a sequence of tokens they solve the problem of predicting the following token, repeating sequentially until some final token is found.

These architectures leverages autoregressive attention (Vaswani et al. 2017), where each token attends to its preceding tokens, maintaining the temporal order of the sequence. This approach allows them to generate coherent and contextually relevant text by sampling from this learned distribution, while also capturing long-range dependencies and complex patterns in language.

The Llama models (Dubey et al. 2024) use this architecture and fine-tune it to tasks of instruction-following, and are specially adept at complex prompts.

The Decoder-only model **Meta-Llama-3.1-8B-Instruct** has 8 billion parameters, while **Meta-Llama-3.1-70B-Instruct** has 70 billion. Meta provides an even larger model, **Meta-Llama-3.1-405B-Instruct**, that is too large to easily experiment with. However, we expect the difference between this model and the 70B model to be roughly the same as the difference between the 70B and 8B models.

The properties of the models are summarised in Table 3.

Model	Type	Developer	Parameters	Context
Flan-T5-XL	Seq2Seq	Google Brain	3 Billion	512 Tokens
Flan-T5-XXL	Seq2Seq	Google Brain	11 Billion	512 Tokens
Meta-Llama-3.1-8B-Instruct	Decoder-Only	Meta AI	8 Billion	128k Tokens
Meta-Llama-3.1-70B-Instruct	Decoder-Only	Meta AI	70 Billion	128k Tokens

Table 3: Model cards of the large language models used in this research.

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 which is given by the original query. An example of this shuffling is in Table 5.

Later, we add this *counterparametric answer* to the context, to form a new query and query the same model again; this is exemplified in Table 4.

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. While beam search with a short beam width tends to produce more

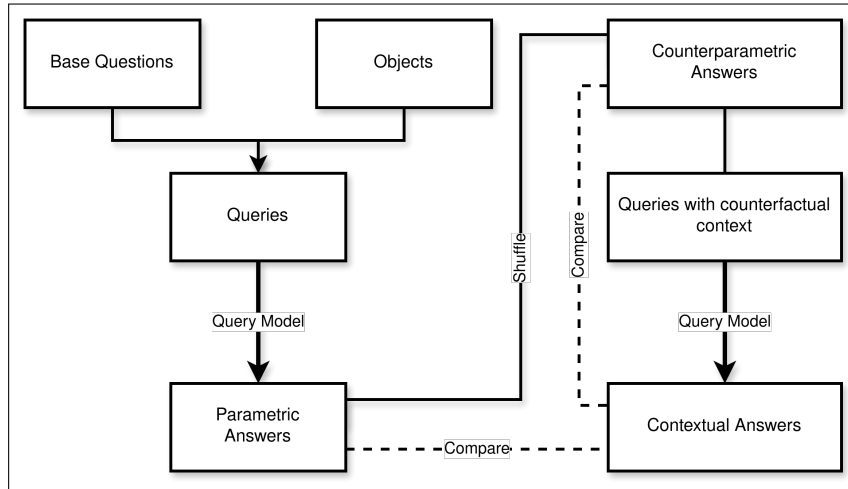


Figure 3: Example diagram of steps used to calculate the two sets of answers, *parametric* and *contextual*, and to compare them to answer the question in this objective. Many of the terms in this diagram are explained in the Glossary.

accurate results for long answers (Sutskever et al. 2014, Wu et al. 2016) and there are many other sampling methods that produce better results (Holtzman et al. 2020), this is likely to not have an effect on experiments that result in shorter answers (Raffel et al. 2020).

We compare the parametric answer to the previous values to come to one of three cases:

Parametric answers are equal to the answer given by the model when queried without context. This answer would come from the parametric memory of the model, and could potentially include hallucinations not fixed by RAG.

Contextual answers are equal to the context given in the query.

Other answers are neither of these, and this answer comes from a mis-interpretation of the input by the model or from some other source.

To minimise the amount of problems caused by large language models generating extra information, we define two strings to be equal by removing all text after a period or an `<EOS>` token, removing punctuation and stop words, and finding whether one is a subsequence of another.

This approach to compare strings is not enough, and understanding whether two answers are identical is an ongoing problem. This is explained in the Future Work section in Section 6.1.1.

The approach to finding out the source of knowledge of each query is specified in Figure 3.

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.

Base Question	Object	Parametric Answer	Counterparametric Answer	Question with Counterparametric Context
	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
Q: What is the date of birth of {person}? A: The date of birth of {person} 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 5: Using the same question format allows us to repurpose previous parametric answers as counterparametric ones.

3.3 Studying the Role of Perplexity Scores of the Provided Answers

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 6.

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

		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 6: 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, \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.

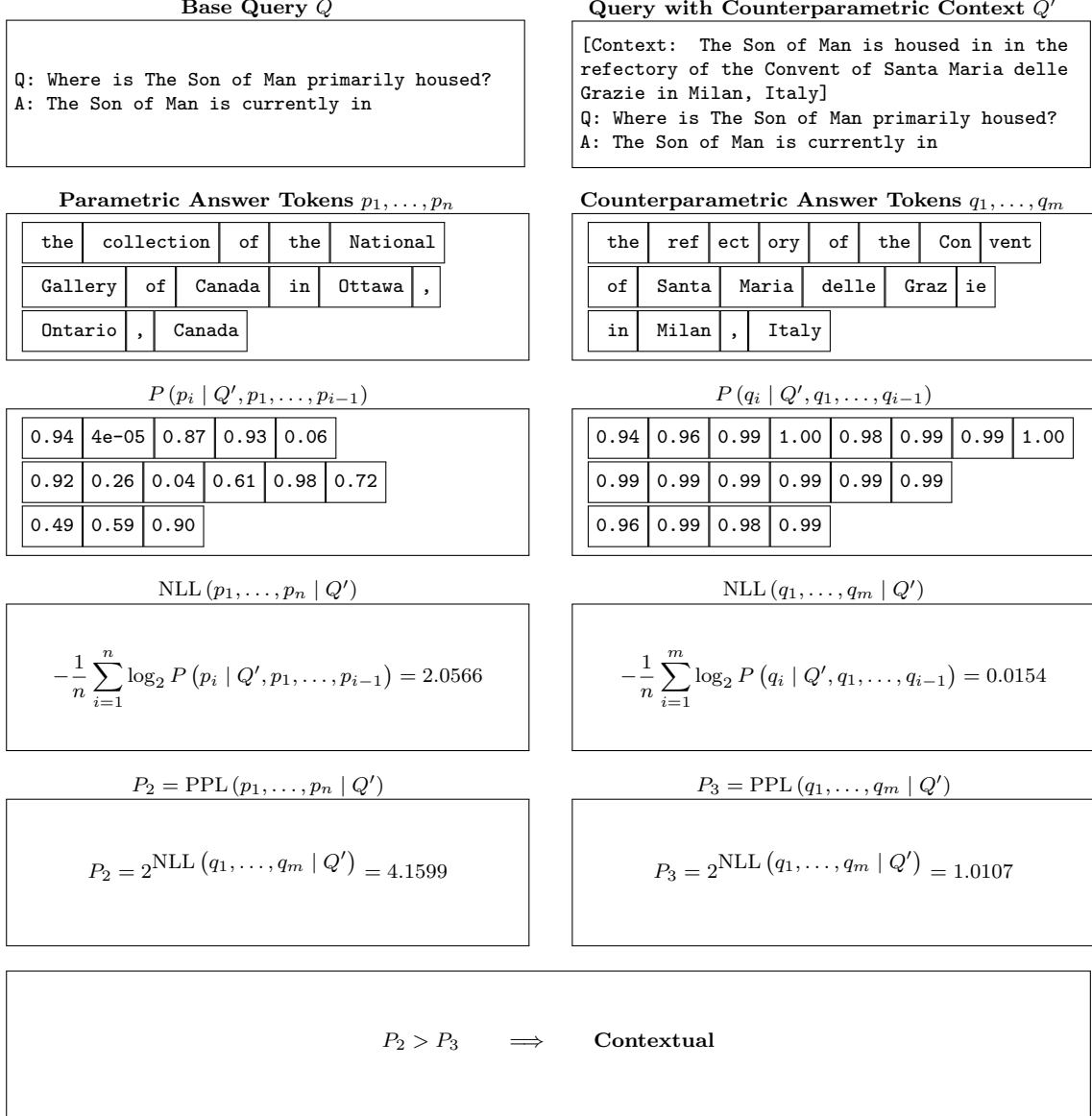


Figure 4: Example of perplexity calculation for the parametric and counterparametric answers in a query with the counterparametric context. Note that, due to teacher forcing, the calculation finds the probability of the next token p_i given the previous tokens of the searched answer p_1, \dots, p_{i-1} 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 3.1.

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 7.

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 7: 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 3.1 with added counterparametric context on each of the four models the four models can be found in Table 8 and Figure 5.

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

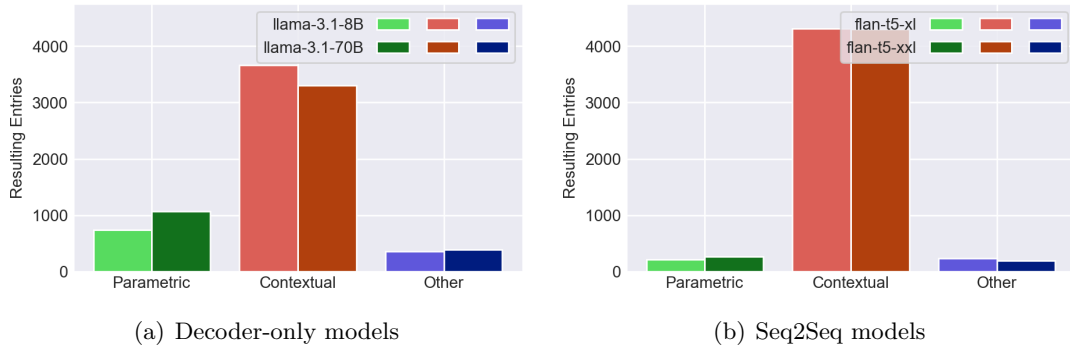


Figure 5: Results by type depending on which model; these are the same results as Table 8.

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 9 and 10 and Figures 6 and 7.

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



Figure 6: Results of running decoder-only models on the queried data, grouped by category. This plots the information shown in Table 9.

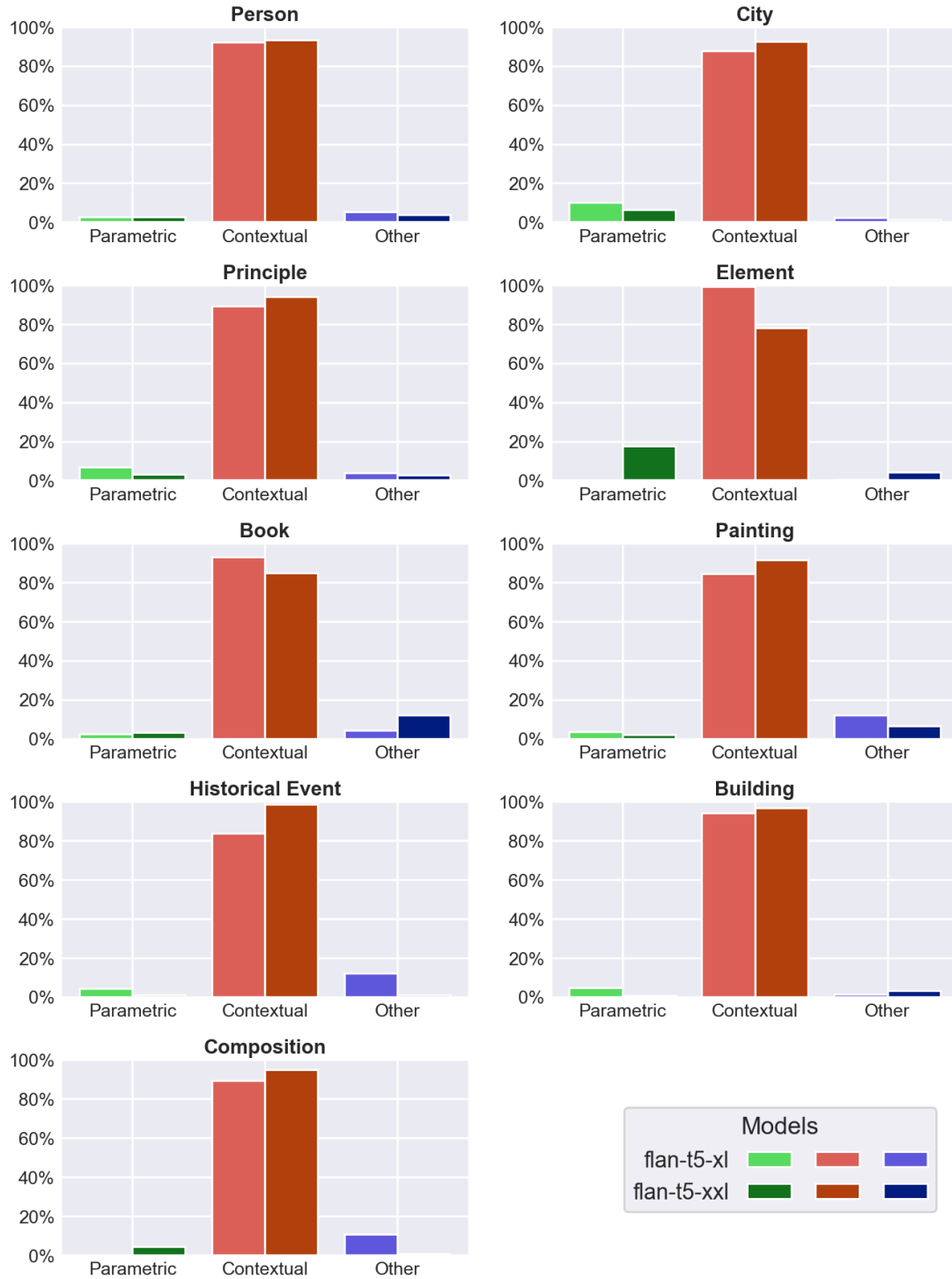


Figure 7: Results of running Seq2Seq models on the queried data, grouped by category. This plots the information shown in Table 10.

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 11 and 12 and Figure 8.

	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 11: 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 12: Distribution of perplexity values for Seq2Seq models

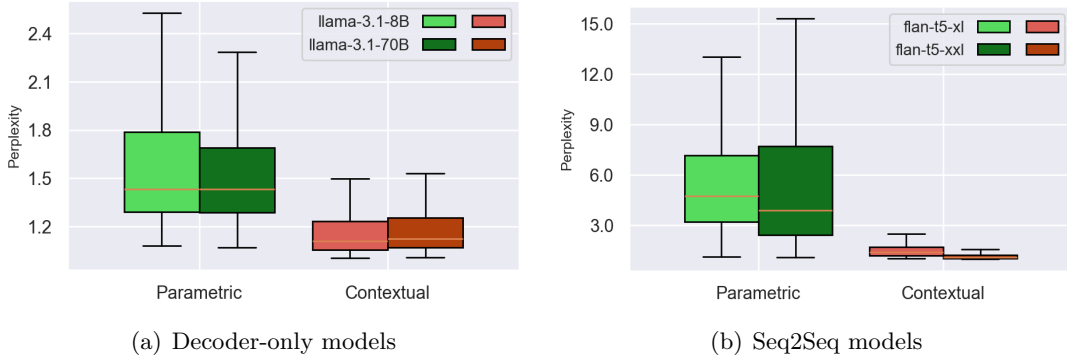


Figure 8: Perplexity distribution according to model type and size. These represent the same distributions as Tables 11 and 12.

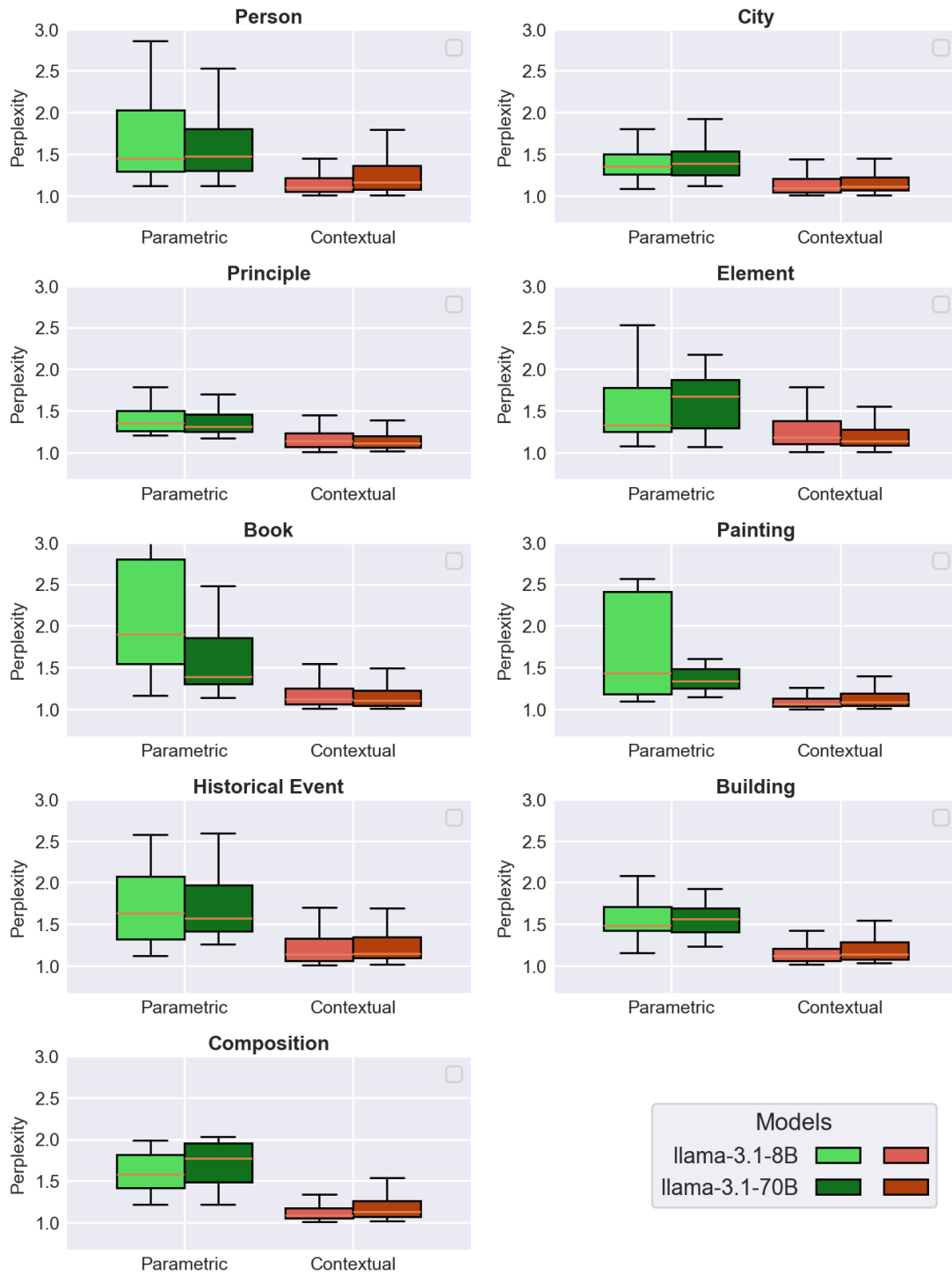


Figure 9: Box plots representing the distribution of the perplexities when running both Llama models, grouped by category.

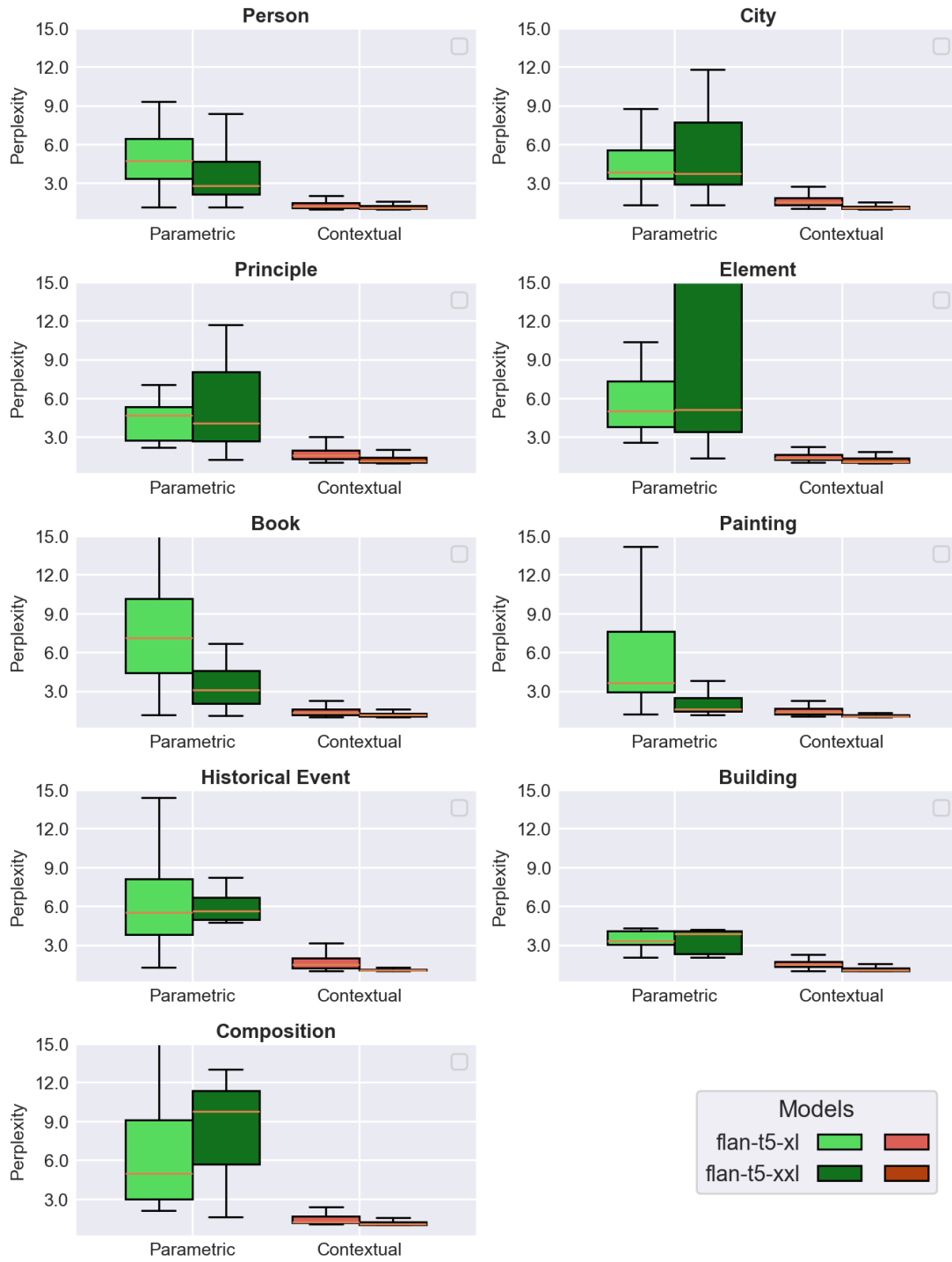


Figure 10: Box plots representing the distribution of the perplexities when running both Flan-T5 models, grouped by category.

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 13 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 13: 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>
- Holtzman, A., Buys, J., Du, L., Forbes, M. & Choi, Y. (2020), ‘The curious case of neural text degeneration’, *arXiv preprint arXiv:1904.09751*.
- 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>
- Norouzi, M., Mikolov, T., Bengio, S., Singer, Y., Shlens, J., Frome, A., Corrado, G. S. & Dean, J. (2014), ‘Zero-shot learning by convex combination of semantic embeddings’.
URL: <https://arxiv.org/abs/1312.5650>
- Radford, A. & Narasimhan, K. (2018), Improving language understanding by generative pre-training.
URL: <https://api.semanticscholar.org/CorpusID:49313245>
- Radford, A., Wu, J., Child, R., Luan, D., Amodei, D. & Sutskever, I. (2019), ‘Language models are unsupervised multitask learners’, *OpenAI blog* **1**(8), 9.
- 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.
- Sutskever, I., Vinyals, O. & Le, Q. V. (2014), ‘Sequence to Sequence Learning with Neural Networks’.
URL: <https://arxiv.org/abs/1409.3215>
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. & Polosukhin, I. (2017), Attention is all you need, *in* ‘Proceedings of the 31st International Conference on Neural Information Processing Systems’, NIPS’17, Curran Associates Inc., Red Hook, NY, USA, p. 6000–6010.
- Whitehouse, C., Chamoun, E. & Aly, R. (2023), ‘Knowledge Grounding in Retrieval-Augmented LM: An Empirical Study’, *arXiv preprint* .
- Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., Klingner, J., Shah, A., Johnson, M., Liu, X., Łukasz Kaiser, Gouws,

S., Kato, Y., Kudo, T., Kazawa, H., Stevens, K., Kurian, G., Patil, N., Wang, W., Young, C., Smith, J., Riesa, J., Rudnick, A., Vinyals, O., Corrado, G., Hughes, M. & Dean, J. (2016), ‘Google’s Neural Machine Translation System: Bridging the Gap between Human and Machine Translation’.

URL: <https://arxiv.org/abs/1609.08144>

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 2: All base questions used in this work. Each one of these will get combined with data from Listing 2 as detailed in Section 3.1.

```

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 3: 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 3.1, 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('--runs-per-question', type = int, default = 1, help =
43     'How many runs (with random counterfactuals) to do for each question.')
44
45     parser.add_argument('base_questions_file', type = open, help = 'File with
46     questions')
47     parser.add_argument('things_file', type = open, help = 'File with things to
48     combine')
49
50     args = parser.parse_args()
51
52     args.base_questions = [x.strip() for x in args.base_questions_file if any(not
53     y.isspace() for y in x)]
```

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

```

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

```

```

96         model_filename = os.path.join(args.output_dir, model + '.csv')
97         with open(model_filename, 'w') as out:
98             print_parametric_csv(out, model_answers)
99
100         elif args.per_model:
101             print_parametric_csv(sys.stdout, model_answers)
102         else:
103             answers |= model_answers
104
105     if answers:
106         logging.info('Writing CSV')
107         print_parametric_csv(sys.stdout, answers)
108
109 if __name__ == '__main__':
110     args = parse_args()
111     if not args.debug:
112         main(args)
113     else:
114         with ipdb.launch_ipdb_on_exception():
115             main(args)

```

Listing 4: `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     runs_per_question: int
31     llm: Model
32
33     def __init__(
34         self,
35         model: Union[str, Model],
36         device: str = 'cpu',
37         max_length: Optional[int] = None,

```

```

37     max_batch_size: Optional[int] = None,
38     runs_per_question: Optional[int] = None,
39 ):
40     self.device = device
41     self.max_length = max_length or 100
42     self.max_batch_size = max_batch_size or 120
43     self.runs_per_question = runs_per_question or 1
44
45     if type(model) == str:
46         model = Model.fromName(model, device = device)
47
48     model = typing.cast(Model, model)
49     self.llm = model
50
51     # Generated list of stop tokens: period, newline, and various different
52     end tokens.
53     stop_tokens = {'.', '\n'}
54     self.stop_token_ids = torch.tensor([
55         v
56         for k, v in self.llm.tokenizer.get_vocab().items()
57         if
58             k in ['<start_of_turn>', '<end_of_turn>',
59                 self.llm.tokenizer.special_tokens_map['eos_token']] or
60             not stop_tokens.isdisjoint(self.llm.tokenizer.decode(v))
61     ]).to(self.device)
62
63     # Query data related to a list of questions, and return a dict with
64     information about these runs.
65     # Output elements:
66     # parametric: Parametric answer, as a string.
67     # base_proba: Perplexity of parametric answer in base query.
68     # counterfactual: Randomly selected counterfactual answer.
69     # base_cf_proba: Perplexity of counterfactual answer in base query.
70     # contextual: Contextual answer, as a string.
71     # ctx_proba: Perplexity of contextual answer.
72     # ctx_param_proba: Perplexity of parametric answer when running contextual
73     query.
74     # ctx_cf_proba: Perplexity of counterfactual answer when running contextual
75     query.
76     # comparison: Comparison between parametric and contextual answer. Where
77     does this answer come from?
78     # preference: Comparison between perplexity of parametric and counterfactual
79     answer on contextual query. Which one is the least surprising?
80     def answerChunk(self, questions: list[Question]) -> dict[str, Any]:
81         output: defaultdict[str, list[Any]] = defaultdict(lambda: [])
82
83         base_tokens = self.tokenise([q.format(prompt = self.llm.prompt) for q in
84             questions])
85         parametric = self.generate(base_tokens)
86
87         parametric_output = self.decode(parametric)
88         base_proba_output = self.perplexity(base_tokens, parametric)
89         for run in range(self.runs_per_question):
90             run_output: dict[str, list[Any]] = dict(
91                 parametric = parametric_output,
92                 base_proba = base_proba_output,
93             )
94
95             run_output['question'] = questions
96             flips = sample_counterfactual_flips(questions,
97                 run_output['parametric'])
98             counterfactual = parametric[flips]

```

```

90         run_output['counterfactual'] = self.decode(counterfactual)
91         run_output['base_cf_proba'] = self.perplexity(base_tokens,
92 counterfactual)
93
94         run_output |= self.answerCounterfactuals(questions,
run_output['counterfactual'], parametric, counterfactual)
95
96         run_output['comparison'] = [
97             'Parametric' if self.streq(a, p) else
98             'Contextual' if self.streq(a, c) else
99             'Other'
100         for p, c, a in zip(run_output['parametric'],
run_output['counterfactual'], run_output['contextual'])
101     ]
102
103         run_output['preference'] = [
104             'Parametric' if pp > cp else
105             'Contextual'
106         for pp, cp in zip(run_output['ctx_proba'],
run_output['ctx_cf_proba'])
107     ]
108
109         for k, v in run_output.items():
110             output[k].extend(v)
111
112         return output
113
114     # Given a list of questions with assigned counterfactuals, run contextual
115     # queries and return
116     # a dictionary containing information about these runs.
117     # Parameter list:
118     #   questions: list of questions to ask.
119     #   counterfactuals: counterfactual answers, as string.
120     #   parametric: parametric answer, as set of tokens.
121     #   This will be used to calculate the perplexity of this answer with the
122     #   counterfactual context.
123     #   counterfactual: counterfactual answers, as a set of tokens.
124     #   This is necessary since the same string might have several encodings,
125     #   but we need exactly the same one generated by the model
126     #   in the first place.
127     def answerCounterfactuals(self, questions: list[Question], counterfactuals:
list[str], parametric: LongTensor, counterfactual: LongTensor) -> dict[str,
Any]:
128         output: dict[str, Any] = {}
129         ctx_tokens = self.tokenise([
130             q.format(prompt = self.llm.cf_prompt, context = context)
131             for q, context in zip(questions, counterfactuals)
132         ])
133         contextual = self.generate(ctx_tokens)
134
135         output['contextual'] = self.decode(contextual)
136         output['ctx_proba'] = self.perplexity(ctx_tokens, contextual)
137
138         output['ctx_param_proba'] = self.perplexity(ctx_tokens, parametric)
139         output['ctx_cf_proba'] = self.perplexity(ctx_tokens, counterfactual)
140
141         return output
142
143     # Answer a list of Questions: run the queries, gather counterfactual values,
144     # run the queries

```

```

142 # with counterfactual context, and return a 'dict' with information to print.
143 @torch.no_grad()
144 def answerQueries(self, questions: list[Question]) -> dict[str, Any]:
145     output: defaultdict[str, list[Any]] = defaultdict(lambda: [])
146
147     chunks = chunk_questions(questions, max_batch_size = self.max_batch_size)
148     logging.info(f'Answering {len(questions)} queries in {len(chunks)}
149     chunks.')
```

```

149
150     for e, chunk in enumerate(chunks, start = 1):
151         logging.info(f'Parsing chunk ({e} / {len(chunks)}), which has size
152         {len(chunk)}.', extra = {'rate_limit': 20})
153
154         chunk_output = self.answerChunk(chunk)
155
156         for k, v in chunk_output.items():
157             output[k] += v
158
159     return dict(output)
160
161 # Tokenise a list of phrases.
162 # [n] -> (n, w)
163 def tokenise(self, phrases: list[str]) -> BatchEncoding:
164     return self.llm.tokenizer(
165         phrases,
166         return_tensors = 'pt',
167         return_attention_mask = True,
168         padding = True,
169     ).to(self.device)
170
171 # Generate an attention mask for a sequence of tokens.
172 # (n, w) -> (n, w)
173 def batch_encode(self, tokens: LongTensor) -> BatchEncoding:
174     attention_mask = tokens != self.llm.tokenizer.pad_token_id
175     return BatchEncoding(dict(
176         input_ids = tokens,
177         attention_mask = attention_mask,
178     ))
179
180 # Use Greedy decoding to generate an answer to a certain query.
181 # (n, w) -> (n, w)
182 def generate(self, query: BatchEncoding) -> LongTensor:
183     generated = self.llm.model.generate(
184         input_ids = query.input_ids,
185         attention_mask = query.attention_mask,
186         max_new_tokens = self.max_length,
187         min_new_tokens = self.max_length,
188         tokenizer = self.llm.tokenizer,
189         do_sample = False,
190         temperature = None,
191         top_p = None,
192         return_dict_in_generate = True,
193         pad_token_id = self.llm.tokenizer.pad_token_id,
194         eos_token_id = self.llm.tokenizer.eos_token_id,
195         bos_token_id = self.llm.tokenizer.bos_token_id,
196     )
197
198     # Ensure that all the sequences only contain <PAD> after their first stop
199     # token.
200     sequences = generated.sequences[:, -self.max_length:]
201     ignores = torch.cumsum(torch.isin(sequences, self.stop_token_ids), dim =
202     1) > 0

```



```

200     sequences[ignores] = self.llm.tokenizer.pad_token_id
201
202     return sequences
203
204     # Return the perplexity of a certain sequence of tokens being the answer to a
205     # certain query, as a list of floats in CPU.
206     # (n, w0), (n, w1) -> (n)
207     def perplexity(self, query: BatchEncoding, answer: LongTensor) -> list[float]:
208         probs = self.batch_perplexity(query, self.batch_encode(answer))
209         return probs.cpu().tolist()
210
211     # Return the perplexity of a certain sequence of tokens being the answer to a
212     # certain query.
213     # (n, w0), (n, w1) -> (n)
214     @torch.no_grad()
215     def batch_perplexity(self, query: BatchEncoding, answer: BatchEncoding) ->
216     FloatTensor:
217         entropies = self.llm.logits(query, answer).log_softmax(dim = 2)
218         entropies /= math.log(2)
219         probs = torch.where(
220             answer.input_ids == self.llm.tokenizer.pad_token_id,
221             torch.nan,
222             entropies.gather(index = answer.input_ids.unsqueeze(2), dim =
223             2).squeeze(2),
224         )
225
226         return torch.pow(2, -torch.nanmean(probs, dim = 1))
227
228     # Decode a sequence of tokens into a list of strings.
229     # (n, w) -> [n]
230     def decode(self, tokens: LongTensor) -> list[str]:
231         decoded = self.llm.tokenizer.batch_decode(
232             tokens,
233             skip_special_tokens = True,
234             clean_up_tokenization_spaces = True,
235         )
236         return [x.strip() for x in decoded]
237
238     # Compare strings for equality to later check whether an answer is parametric
239     # or contextual.
240     # For simplicity, we remove stop words and gather only the subset of words.
241     @staticmethod
242     def streq(a: str, b: str) -> bool:
243         a = a.lower().replace('the', '').replace(',', '').strip()
244         b = b.lower().replace('the', '').replace(',', '').strip()
245         return a[:len(b)] == b[:len(a)]

```

Listing 5: 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 = {

```

```

10     'llama': 'meta-llama/Meta-Llama-3.1-8B-Instruct',
11     'llama-70b': 'meta-llama/Meta-Llama-3.1-70B-Instruct',
12     'llama-405b': 'meta-llama/Meta-Llama-3.1-405B-Instruct',
13     'flan-t5': 'google/flan-t5-base',
14     'flan-t5-small': 'google/flan-t5-small',
15     'flan-t5-base': 'google/flan-t5-base',
16     'flan-t5-large': 'google/flan-t5-large',
17     'flan-t5-xl': 'google/flan-t5-xl',
18     'flan-t5-xxl': 'google/flan-t5-xxl',
19     'gemma': 'google/gemma-2-9b-it',
20     'gemma-27b': 'google/gemma-2-27b-it',
21     'falcon2': 'tiiuae/falcon-11b',
22     'falcon-180b': 'tiiuae/falcon-180b-chat',
23     'falcon-40b': 'tiiuae/falcon-40b-instruct',
24     'falcon-7b': 'tiiuae/falcon-7b-instruct',
25     'distilbert': 'distilbert/distilbert-base-uncased-distilled-squad',
26     'roberta': 'FacebookAI/roberta-base',
27     'roberta-large': 'FacebookAI/roberta-large',
28     'roberta-squad': 'deepset/roberta-base-squad2',
29     'mixtral': 'mistralai/Mixtral-8x22B-Instruct-v0.1',
30     'dummy': '',
31 }
32
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
formatting.'
74     # self.cf_prompt = 'Answer the following question using the previous
context in a few words and with no formatting.'
75     self.prompt = ''
76     self.cf_prompt = ''
77
78     kwargs = {}
79     if 'llama' in name:
80         kwargs = dict(
81             pad_token = '<|reserved_special_token_0|>',
82             padding_side = 'left',
83         )
84     elif 'gemma' in name:
85         kwargs = dict(
86             padding_side = 'right',
87         )
88
89     self.tokenizer = AutoTokenizer.from_pretrained(
90         self.model_name,
91         clean_up_tokenization_spaces = True,
92         **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),
length, self.tokenizer.pad_token_id)
158
159         return self.model(
160             input_ids = input_ids,
161             attention_mask = attention_mask,
162             decoder_input_ids = decoder_input_ids,
163         ).logits[:, : answer.input_ids.shape[1]]
164
165     # Large decoder-only model.
166     # Similar to DecoderOnlyModel, but eagerly deletes the model when the class is
deleted.
167     # Assumes you need 2 GPUs to run this.
168     class LargeDecoderOnlyModel(DecoderOnlyModel):
169         def __init__(self, name: str = 'cuda'):
170             if torch.cuda.device_count() < 2:
171                 raise ValueError(f'At least two GPUs are needed to run {name}')
172
173             super().__init__(name, device)
174
175         def __del__(self):
176             logging.info(f'Deleting large model {self.name}')
177             del self.model
178             torch.cuda.empty_cache()
179
180     # Dummy model, used for testing.
181     class DummyModel(Model):
182         def __init__(self):
183             nn.Module.__init__(self)
184             self.name = 'dummy'
185             self.tokenizer = self
186             self.model = self
187             self.sequences = ['dummy']

```

```

188         self.logits = tensor([[[1., 2., 3.]])
189
190         self.bos_token_id = 0
191         self.eos_token_id = 1
192         self.pad_token_id = 2
193
194     def to(self, *args, **kwargs):
195         return self
196
197     def __call__(self, *args, **kwargs):
198         return self
199
200     def generate(self, *args, **kwargs):
201         return self
202
203     def __getitem__(self, key):
204         return self
205
206     def decode(self, *args, **kwargs):
207         return 'Dummy text'
208
209     def batch_decode(self, *args, **kwargs):
210         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 6: 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, Any
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         current_time = time.time()
23         if current_time - self.last_log[record.lineno] >= record.rate_limit:
24             self.last_log[record.lineno] = current_time
25             return True
26
27         return False
28
29 # A question contains combines a base_question and an object into something that
   can be queried.
30 @dataclass
31 class Question:
32     category: str
33     obj: str
34     base_question: str
35
36     # Static constructor: return a question combining an object and an object if
   the category
37     # matches; return None otherwise.
38     @staticmethod
39     def orNothing(obj: str, category: str, base_question: str) ->
   Optional['Question']:
40         if not f'{{{category}}}' in base_question:
41             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             q = Question.orNothing(obj = obj['object'], category =
   obj['category'], base_question = bq)
71             if q is None:
72                 continue
73
74             questions.append(q)

```

```

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)),
80 lim_questions))}
81     short_questions = [questions[x] for x in keep_nums.keys()]
82
83     return short_questions
84
85 # Given a list of questions and a list of answers, produce a list of integers
86 # that would provide the
87 # index to a randomly sampled counterfactual.
88
89 def sample_counterfactual_flips(questions: list[Question], answers: list[str]) ->
90 list[int]:
91     flips = [-1 for _ in questions]
92
93     for q, es_iter in itertools.groupby(range(len(questions)), key = lambda e:
94 questions[e].base_question):
95         es = set(es_iter)
96
97         for e in es:
98             rest = [x for x in es if answers[x] != answers[e]]
99             if not rest:
100                 logging.error(f'Unitary question "{q}". This means that all
101 answers in this chunk are identical, and the results will be incorrect.')
102                 flips[e] = e
103                 continue
104
105                 flips[e] = random.choice(rest)
106                 assert answers[flips[e]] != answers[e]
107
108     assert all(x != -1 for x in flips)
109     return flips
110
111 # Chunk a list of question into batches of size or at most 'max_batch_size'.
112
113 def chunk_questions(questions: list[Question], max_batch_size: int) ->
114 list[list[Question]]:
115     result: list[list[Question]] = []
116
117     for q, chunk_iter in itertools.groupby(questions, key = lambda x:
118 x.base_question):
119         chunk = list(chunk_iter)
120         if not result or len(chunk) + len(result[-1]) > max_batch_size:
121             result.append([])
122
123             result[-1].extend(chunk)
124
125     return result
126
127 # Prints a CSV file with the questions and resulting answers.
128
129 def print_parametric_csv(out: typing.TextIO, answer: dict[str, list[Any]]):
130     fieldnames = ['Num', 'Category', 'Base_Question', 'Thing', 'Question',
131 'Prefix'] + list(answer.keys())
132
133     writer = csv.DictWriter(
134         out,
135         fieldnames = fieldnames,
136         extrasaction = 'ignore',
137         dialect = csv.unix_dialect,
138         quoting = csv.QUOTE_MINIMAL,
139     )

```

```

129     writer.writeheader()
130
131     for e, answers in enumerate(zip(*answer.values())):
132         param = dict(zip(answer.keys(), answers))
133
134         question = param['question']
135         param.pop('question')
136
137         writer.writerow(
138             {
139                 'Num': str(e),
140                 'Category': question.category,
141                 'Base_Question':
142                 ''.join(question.base_question.partition('?')[0:2]),
143                 'Object': question.obj,
144                 'Question': question.format(use_later = False),
145                 'Prefix': question.format(use_question = False)
146             } | param

```

Listing 7: 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
18 string', ''])
19     )
20
21     def test_winner(self):
22         logits = tensor([
23             [[0.0900, 0.2447, 0.6652], [0.6652, 0.2447, 0.0900], [0.2447, 0.6652,
24 0.0900]],
25             [[0.2119, 0.2119, 0.5761], [0.2119, 0.2119, 0.5761], [0.2119, 0.2119,
26 0.5761]],
27             [[0.6652, 0.2447, 0.0900], [0.2119, 0.5761, 0.2119], [0.5761, 0.2119,
28 0.2119]],
29         ])
30
31         expected_path = tensor([[2, 0, 1], [2, 2, 2], [0, 1, 0]])
32         expected_probs = tensor([
33             [0.6652, 0.6652, 0.6652],
34             [0.5761, 0.5761, 0.5761],
35             [0.6652, 0.5761, 0.5761],
36         ])
37
38         path, probs = self.qa.winner(logits)
39         self.assertTrue(torch.equal(path, expected_path), msg = (path,

```



```

36         expected_path))
37         self.assertTrue(torch.allclose(probs, expected_probs), msg = (probs,
38         expected_probs))
39
40     def test_decode(self):
41         path = tensor([
42             [128000, 9906, 1268, 527, 499, 13, 358, 1097, 3815, 7060, 9901,
43             499, 13],
44             [128000, 3648, 1074, 1618, 198, 54953, 0, 13, 1234, 1234, 1234,
45             1234, 1234],
46             [128000, 2822, 3009, 925, 1234, 1234, 1234, 1234, 1234, 1234, 1234,
47             1234, 1234],
48             [128000, 13, 1234, 1234, 1234, 1234, 1234, 1234, 1234, 1234, 1234,
49             1234, 1234],
50         ])
51         probs = tensor([
52             [1., 3., 5., 7., 9., 11., 13., 15., 17., 19., 21., 23., 25.],
53             [1., 3., 5., 7., 9., 11., 13., 15., 17., 19., 21., 23., 25.],
54             [1., 3., 5., 7., 9., 11., 13., 15., 17., 19., 21., 23., 25.],
55             [1., 3., 5., 7., 9., 11., 13., 15., 17., 19., 21., 23., 25.],
56         ])
57
58         expected_result = [
59             'Hello how are you',
60             'Newline here',
61             'No stop string',
62             '',
63         ]
64         expected_mean_probs = [5., 4., 13., 1.]
65
66         result, mean_probs = self.qa.decode(path, probs)
67         self.assertEqual(expected_result, result)
68         self.assertEqual(expected_mean_probs, mean_probs)

```

Listing 8: `test.QuestionAnswerer.py` is used to test some of the complicated bits of logic in `QuestionAnswerer`.