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

I wanted to express my gratitude to my supervisor, Dr. Tillman Weyde (with apologies for getting his first name wrong for months), to Dr. Chenxi Whitehouse, the original author of the preprint, and to Dr. Pranava Madhyastha for helping me take this great step in my academic career. This was not an easy thesis to research and write, but the guidance and advice I received from each of the three collaborators helped me fully understand the context I was working on and in conducting meaningful research.

My most important takeaway from the thesis, which I learned thanks to the work from my collaborators, is my deeper understanding of how to conduct a rigorous academic project. I plan to pursue a doctorate degree in the near future, and I am sure that the knowledge I gained in this project will make my future projects more thorough and efficient, and greatly improve my future academic work.

I also wanted to acknowledge the work done by the entire MSc in Artificial Intelligence in City St. Georges, University of London, for making a fantastic course. I learned a lot in many subjects and areas, even in ones which I barely had an idea were so broad. I also made a lot of contacts and good friends, who will definitely continue being a presence in my life after this MSc. Most important of all, I had fun, and any doubts I had on returning to academia after many years in the industry were completely unfounded.

In particular, I wanted to acknowledge Dr. Esther Mondragón for being an outstanding professor and administrator, and for taking a very strong stance in ensuring that the standards for this Master's degree remain high. My experience in this program was fantastic, and I hope the same continues being true for future students.

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 contexts where precision and correctness are 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 problem. However, this solution is far from perfect as it's unclear when a large language model will choose to generate answers using the context provided by RAG over the knowledge in its parametric memory.

This thesis explores the *Knowledge Grounding* of various large language models. In particular, it attempts to answer a research question: **How does a large language model respond when given information that contradicts its inherent knowledge, and why?**

To investigate this, we develop a diverse dataset comprising questions from various topics and globally representative data. We use this dataset to construct queries with counterparametric context across four models of different architectures and sizes, and later we test models of various architectures and sizes to find out which type of answer they choose. We also analyse the *perplexity* of these answers to give us a clue to why the model chose an answer over the other, and to make a rough prediction of the source of a particular answer.

Our findings suggest that, when including the kind of contextual information added by RAG, smaller models and models that encode the entire input sequence into an internal representation before outputting an answer might produce more answers sourced from the RAG-provided context, which is generally less affected by 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, while encoder-decoder Seq2Seq models tend to outperform Decoder-only models.

As an extra analysis we investigate methods for determining whether a given response originates from the RAG context or the model's internal memory from the query's resulting perplexity. This might be used to develop methods to prevent hallucinations in large language models that use RAG indexing.

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

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 answering questions using solely the parametric knowledge from these models might not be the most effective way to solve problems that are not directly related to text generation (Yao et al. 2023).

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

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

Knowledge grounding in a large language model with RAG-generated data can improve its performance. In general, a well-grounded model outputs data that is anchored in verifiable and accurate sources. In the context of queries enhanced with RAG, we prefer knowledge sourced from context which came from the index, since they are much less likely to be "hallucinations" or mistakes from the model.

This project aims to understand the performance of various large language models when queries with added context 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 to improve our understanding of how different LLMs interact with the given context in the problem of question answering. Specifically, we research whether these interactions vary depending on the type of question being answered for models of different architectures and sizes, 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:

How does a large language model respond when given information that contradicts its inherent knowledge, and why?

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. **Creating a representative dataset of questions.**
   This is necessary as existing Q&A datasets are not suitable for our objectives.

2. **Building an experimental framework to understand the source of an LLM's answer.**
   This will give us information about which models chooses to answer questions using its learned parametric knowledge or the given context, and whether it depends on the question asked.

3. **Enhancing the framework to understand the reasoning behind each answer**
   We use the perplexity of a model's response on both answers to understand *why* a certain answer was chosen, and we attempt to predict the source of the answer with this number alone. This can help RAG-enhanced systems prevent hallucinations by repeating the index search in answers which might contradict the given context.

1.4 Overview of Methods

1.4.1 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 whether each response came from the model's parametric memory or from the RAG-provided context, and should be reasonably representative of the world to prevent biases.

In particular, the questions should allow us to easily create counterparametric answers to later add as context to our queries. We follow the example of Yu et al. on creating questions that can be easily answered with short responses, and later using these answers to create counterparametric context.

We enhance the work by Yu et al. by adding a much larger and broader set of questions from a large variety of topics.

1.4.2 Building an experimental framework to understand the source of an LLM's answer

Little is currently understood about the factors that control the source of knowledge of an answer in a large language model, and whether the generated text comes from the query's context or from memorised parametric information.

Previous research found out that, when the context of a query contradicts the ground knowledge of a model, the final answer is affected by the size and architecture 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.

1.4.3 Enhancing the framework to understand the reasoning behind each answer

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 for morels with access to its ground truth, as is the case in models like Pythia (Biderman et al. 2023).

Unfortunately, most open-source large language models provide only the model weights and 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 came from a large language model's memory or whether it was derived from the provided context.

2 Context

This research is the latest on a long line of academic articles on retrieval-augmented generation, comparing and contrasting parametric knowledge with contextual information, 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 transformer architecture (Vaswani et al. 2017). This architecture relies entirely on self-attention mechanisms rather than recurrent 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.

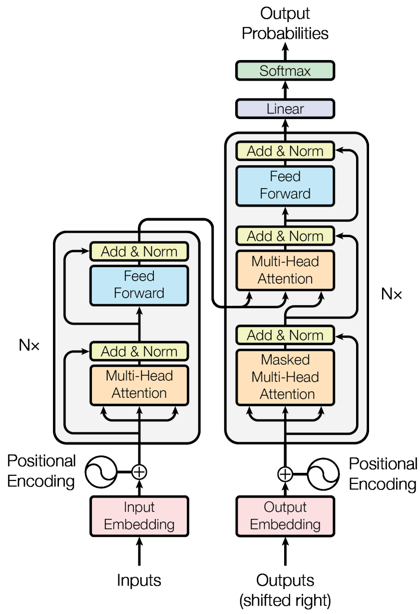


Figure 1: Transformer Architecture, from "Attention is all you need" (Vaswani et al. 2017).

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 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).

Despite these improvements in sequential models, the models still present many faults in question-and-answer tasks (Jiang et al. 2021). Hallucinations are not uncommon, even for queries that ask questions that can be answered with short answers.

2.2 Architectures of Large Language Models

In this subsection we present two different large language model architectures that are used widely for this research. This thesis uses and compares four models using the Flan-T5 and Llama architectures.

2.2.1 Seq2Seq Models: T5 and Flan-T5

The T5 model (Raffel et al. 2020), developed by Google Brain in 2020, is an encoder-decoder Seq2Seq model that treats every processing problem as a text-to-text task. This way it uses transfer learning to learn many different kinds of tasks from a single source of training data.

The model is trained using only masked language modelling and treating every problem as a supervised text-to-text task where both the input and output are in text form. The data is later fine-tuned to make it able to solve other tasks.

`Flan-T5` uses the same architecture as T5, but fine-tuned on tasks to follow explicit instructions (Chung et al. 2022). This scales well on larger tasks and larger model sizes, and it's particularly effective on problems requiring a chain-of-thought.

These models have strong performance while adding little or not existing context, and are capable capable to understand explicit instructions which can generalise to new task types, improving the downstream task performance (Longpre et al. 2023).

2.2.2 Decoder-only Models: GPT and Llama

GPT models use a transformer-based decoder-only architecture trained to solve the problem of causal language modelling, or predicting the next token in a sequence (Radford & Narasimhan 2018, Radford et al. 2019, Brown et al. 2020). These models generate text by predicting tokens one-by-one rather than encoding internal information of a query, such as Seq2Seq models.

GPT models are good at solving a variety of tasks they they not trained in with little or no context. Its autoregressive nature specialised it at generating human-like text, so it's widely used for chatbots and content creation.

Llama models improve several limitations of GPT by training smaller models on more data (Touvron et al. 2023), which generally achieves better performance (Hoffmann et al. 2022), and various improvements to the transformer architecture.

These models outperform GPT models on most benchmarks, despite being considerably smaller, by prioritising training efficiency and parameter optimisation. Further improvements to later versions of this model include fine-tuning in a post-training round using reinforcement learning from human feedback and training with longer contexts (Martin et al. 2023), and adding multilingual data to the training data (Dubey et al. 2024).

Llama 3 includes a set of `Instruct` models, which like `Flan-T5` are fine-tuned to following instructions.

One of the inherent limitations of Decoder-only architectures is the challenge of maintaining coherence over long contexts. Additionally, they struggle with using accurate data from the query when it contradicts its parametric knowledge.

2.3 Retrieval-Augmented Generation

Large pre-trained language models store factual knowledge in their parameters. Their ability to access factual information from their source data without the risk of hallucinating incorrect information is limited, which affects their performance on knowledge-intensive tasks such as question-and-answer problems.

Retrieval-Augmented Generation (RAG) attempts to solve this problem by adding extra non-parametric data in a context gathered from an index with a separately-trained retriever (Lewis et al. 2020). This retrieved context is fed back to the original query as a combined representation containing this extra data. A diagram with an overview of this method is presented in Figure 2.

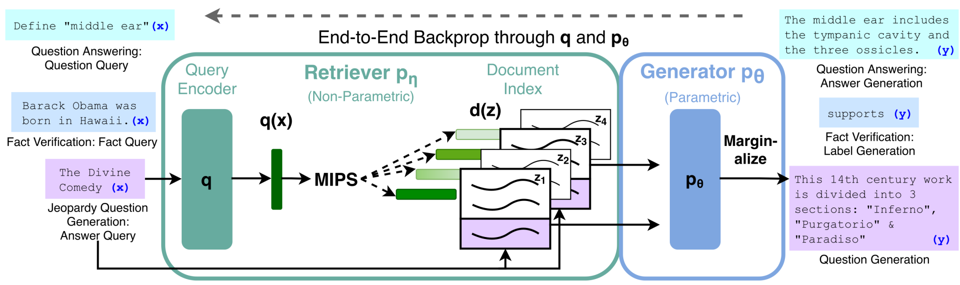


Figure 2: An overview of the RAG approach, combining a pre-trained retriever with a pre-trained model. From "Retrieval-Augmented Generation for Knowledge-Intensive NLP tasks" (Lewis et al. 2020)

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 method is not perfect even after adding contextual data to the query the model could generate a response using its learned parametric memory while ignoring the data retriever by the RAG model.

2.4 Knowledge Grounding on Queries with Added Context

Knowledge grounding is the process by which a large language model incorporates external knowledge into its output generation. In the context of queries with context provided by RAG, a well-grounded model would ensue that answers are consistent with the knowledge in the index rather than in the model's inherent knowledge if they are contradictory.

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 Seq2Seq models such as Flan-T5 have better performance.

"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 architecture and size of the large language model will affect the probability of the model choosing to use the context (which is much less prone to hallucinating) as an answer rather than its parametric knowledge.

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. 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).

```
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

3 Methods

How does a large language model respond when given context that contradicts its parametric, learned knowledge? Why does it choose this answer?

To understand this, we build a new framework for testing a large language model's answers when presented with contradictory information. We test this framework with models of various architectures and sizes to get insights about our responses.

Following the example set in Section 1.3, we split this work into three sub-objectives.

3.1 Creating a representative dataset of questions

As argued in Section 1.4.1, 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 selecting a wide variety of questions questions we ensure that the dataset will provide meaningful insights of the grounding of LLMs across a wide spectrum of domains.

Our dataset should have the following properties.

1. **Questions should have short, unambiguous answers.**
   Our goal is to compare these answers both for equality, on the LLM's perplexity at generating this result. Longer answers make this objective harder to interpret since two long answers might have a higher chance of being both correct. Ensuring our answers are short reduces the space of different but equivalent answers.

2. **Questions must cover a large and diverse set of topics.**
   Parametric answers are sourced from the training data of a large language model, 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 choosing to answer from parametric or contextual knowledge(Yu et al. 2023). We require a large and diverse and set of topics to counteract potential biases.

3. **Questions must allow for the creation of counterparametric answers.**
   In order to test a model's knowledge grounding, this thesis requires understanding whether an answer came from contextual versus inherent knowledge. A simple way to do this is to repeat and enhance the approach used by Yu et al. of adding counterparametric answers to a query context. This allows us to to easily disambiguate whether an answer came from the model's inherent parametric memory or from the given 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 due to 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 question 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 in generating 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 type of question 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 countries' capital queries used in the paper by Yu et al., and creates a similar but larger and more varied dataset of questions and answers from a wide range of topics, assuring questions can be grouped by question pattern so that the formats of their answer are similar. This way, we can emulate the approach used in that paper of reusing the answer from a certain question as the counterfactual context of another.

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

Since this thesis requires a set of questions that covers a large set of topics, we first generate by hand a list of various categories of questions. Each one of these categories refers to the subject of the question, rather than the answer.

For each category we create a set of base questions and another set of objects. We ensure that for all the objects, and specially in the case of people, the resulting list of objects represents a wide variety of objects from across the world

For each category, we generate a set of questions by matching every one of its base questions with every one of its objects. 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. For each category, we match every base question to every object to create a longer set of questions. In this example, the *Person* category contains 2 base questions and 2 objects, resulting in 4 questions; the *City* category contains 1 base question and 4 objects, also resulting in 4 questions.

3.2 Building an experimental framework to understand the source of an LLM's answer

3.2.1 Model Selection

In order to understand the knowledge grounding of a wide variety of large language models, the queries generated in Section 3.1 are tested with four models of different architectures and sizes. These models 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.

All of the models used in this research leverage autoregressive attention using the transformer architecture (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.

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. This is considerably bigger than the base Flan-T5 model (Chung et al. 2022), which will provide better accuracy of its parametric answers.

`Flan-T5-XXL` contains 11 billion parameters, has higher accuracy on the parametric answers as the `XL` model (Chung et al. 2022). However, how the higher amount of parameters will affect its knowledge grounding when running our experiment is still unknown.

Decoder-only models generate answers one token at a time from the input query. Given a sequence of tokens, they generate text one token at a time by attempting to solve the problem of predicting the following token (Radford & Narasimhan 2018).

This thesis uses the `-Instruct` versions of the latest Llama models (Dubey et al. 2024), which use this architecture and fine-tune it to tasks of instruction-following. These models are specially adept at complex prompts. Of the models used in this thesis, `Meta-Llama-3.1-8B-Instruct` has 8 billion parameters, while `Meta-Llama-3.1-70B-Instruct` has 70 billion.

The properties of the models are summarised in Table 3.

Model	**Architecture**	**Trained On**	**Parameters**
`Flan-T5-XL`	Seq2Seq	Public NLP Datasets	3 Billion
`Flan-T5-XXL`	Seq2Seq	Public NLP Datasets	11 Billion
`Meta-Llama-3.1-8B-Instruct`	Decoder-Only	Web text, code, instruction datasets	8 Billion
`Meta-Llama-3.1-70B-Instruct`	Decoder-Only	Web text, code, instruction datasets	70 Billion

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

3.2.2 Understanding the source of the answer in each model

The first step to understanding the knowledge grounding of large language models is to create queries that contain data that contradicts its parametric knowledge 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 done by 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 random sampling can be found in Table 4.

Later, we add this *counterparametric answer* to the context, to form a new query

Base Question	Object	Parametric Answer	Counterparametric Answer	Question with Counterparametric Context
Q: What is the date of birth of {*person*}? A: The date of birth of {*person*} is	Che Guevara	June 14, 1928	965 AD	Context: [the date of birth of Che Guevara is 965 AD]. Q: What is the date of birth of Che Guevara? A: The date of birth of Che Guevara is
	Ibn al-Haytham	965 AD	June 14, 1928	Context: [the date of birth of Ibn al-Haytham is June 14, 1928]. Q: What is the date of birth of Ibn al-Haytham? A: The date of birth of Ibn al-Haytham is
	Boyan Slat	27 January 1994	February 23, 1868	Context: [the date of birth of Boyan Slat is February 23, 1868]. Q: What is the date of birth of Boyan Slat? A: The date of birth of Boyan Slat is
	W.E.B Du Bois	February 23, 1868	June 14, 1928	Context: [the date of birth of W.E.B Du Bois is June 14, 1928]. Q: What is the date of birth of W.E.B Du Bois? A: The date of birth of W.E.B Du Bois is
Q: What country is {*city*} in? A: {*city*} is in	Cairo	Egypt	India	Context: [Cairo is in India]. Q: What country is Cairo in? A: Cairo is in
	Mumbai	India	Egypt	Context: [Mumbai is in Egypt]. Q: What country is Mumbai in? A: Mumbai is in

Table 4: Using the same question format allows us to repurpose previous parametric answers as counterparametric ones. In this example, we randomly sample answers from the same base question but referring to a different object to find counterparametric answers, which we later add to the context of the query.

and query the same model again with the added counterparametric context. This is exemplified in Table 5.

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 generate the answer. While beam search with tends to produce more accurate results for long answers (Sutskever et al. 2014, Wu et al. 2016) and there are many other sampling methods that tend to produce better results (Holtzman et al. 2020), this is likely to not have an effect on experiments shorter answers (Raffel et al. 2020).

We compare the generated answer with the context to the previously generated parametric answer, and we categorise the answer:

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 indicate an hallucination not present in the context.

Contextual answers are equal to the context given in the query. In a RAG context, this would be the answer retrieved from the index.

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 truncate the text until the first period or `<EOS>` token, removing punctuation and stop words, and finding whether one of the answers is a subsequence of another.

We later found out that this approach to compare strings could be improved, and understanding whether two generated answers are identical is an ongoing problem. This is later explained in the Future Work section in Section 6.3.1.

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 5: Example for results for three questions. We enhance each question with a context containing data that is different to the answer given by the model for this question. We later categorise the source of the answer as **Parametric** if it came from the model's inherent memory, **Contextual** if it came from the context, or **Other** if it's neither of these. Note that, in the third query, the model is interpreting the question as asking about Rome in the US State of Georgia, rather than the country of Georgia.

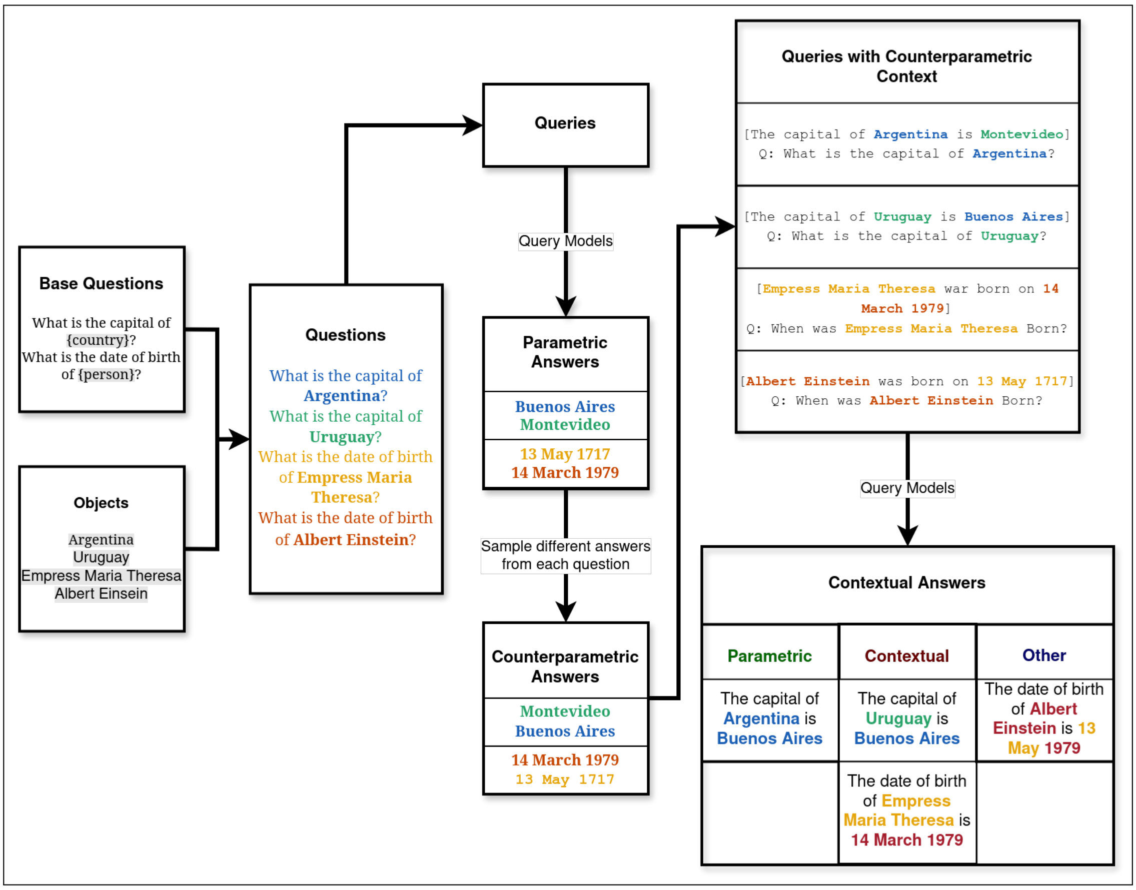


Figure 3: Example diagram of steps taken to calculate the sets of **Parametric**, **Contextual**, and **Other** answers. The terms in this diagram are explained in the Glossary. The following steps are shown:

1. Generate questions by combining every base question with every object of the same category.
2. Query each model, and get a set of parametric answers.
3. Shuffle these parametric answers along the same base question to get counterparametric answers.
4. Query the model again, using the counterparametric answers as context.
5. Group the answers according to their source.

3.2.3 Understanding the result by finding the mean attention of the context and question areas

When comparing different architectures, it's useful to understand how much attention they give to the context compared to the rest of the query.

Given that all our model architectures employ attention mechanisms (Chung et al. 2022, Dubey et al. 2024), we can estimate the relative importance of the tokens in each section of the query by calculating the average self-attention each token receives within its respective section. Specifically, we compute the mean self-attention weights, which serve as a proxy for the emphasis the model places on different parts of the input.

This approach is formalized in Equation (1), where we define the mean self-attention for tokens in the context and query sections.

$$A : \mathbb{R}^{\text{batch} \times |\text{layer}| \times |\text{attn\_head}| \times Q \times K}$$

$$\begin{aligned} m_{b,q,k} &= \frac{1}{|\text{layer}| \times |\text{attn\_head}|} \sum_{\substack{i=0 \\ j=0}}^{\substack{\text{layer} \\ \text{attn\_head}}} A_{b,i,j,q,k} \\ d_{b,q} &= m_{b,q,q} \\ s_{b,i} &= \left(d_{b,i} - \min\left(d_b\right)\right) / \left(\max\left(d_b\right) - \min\left(d_b\right)\right) \end{aligned} \tag{1}$$

$$\text{attn}_{\text{ctx}} = \frac{1}{|\text{ctx}|} \sum_{i \in \text{ctx}} s_{b,i} \qquad \text{attn}_{\text{rest}} = \frac{1}{|\text{rest}|} \sum_{i \in \text{rest}} s_{b,i}$$

In this equation, A is the 5-tensor representing the attention weights. $m_{b,q,k}$ represents the mean attention all layers and heads, while d_b represents the diagonal of these attentions which correspond to the self-attentions.

We normalise the values for the tokens in each query to s_i in order to being able to compare them between different query, and average then among the context tokens and the rest of the query. This results in two results that can be compared between different queries.

3.3 Enhancing the framework to understand the reasoning behind each answer

3.3.1 Perplexity Score

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

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

The probabilities of of the answer generated from a query can be accumulated to calculate the negative log-likelihood NLL, which is used to calculate the perplexity PPL using the formulas from Equations (2) and (3).

$$\mathrm{NLL}\left(x_1, \ldots, x_n \mid Q\right) = -\frac{1}{n} \sum_{i=1}^{n} \log_2 P\left(x_i \mid Q, x_1, \ldots, x_{i-1}\right) \tag{2}$$

$$\mathrm{PPL}\left(x_1, \ldots, x_n \mid Q\right) = 2^{\mathrm{NLL}\left(x_1, \ldots, x_n \mid Q\right)} \tag{3}$$

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

Note that, in these experiments, the token x_n does not necessarily have to be the most likely result generated by the model when applying the query $x_1, \ldots, x_{n-1}$.

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

For a given parametric answer $p_1, \ldots, p_n$ and different counterparametric answer $q_1, \ldots, q_m$, a query without context Q, and a query with this counterparametric context Q' we can calculate four different perplexity scores P_0, P_1, P_2, P_3 as shown in Table 6.

Since the parametric answer is by definition the response of the model to the regular query, $P_0 \leq P_1$ and 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 in a case where the contextual answer is considerably less surprising than the parametric answer.

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

Table 6: We calculate four different perplexity values: one for each set of tokens, and one for each query context. We care about P_2 and P_3, which are the perplexities at getting the parametric and counterfactual answers in a query with counterfactual context.

Similar to the work done earlier in this section, we say that the perplexity of an answer is **Parametric** if the probability of getting the parametric answer is greater than the probability of getting the contextual answer; that is, $P_2 < P_3$. Contrary to this, we consider an answer **Contextual** if $P_2 > P_3$.

For simplicity and to the myriad of possible token and probability combinations, in this experiment we do not analyse the case when an answer is preferred to either of these two.

3.3.3 Predicting whether an answer came from memory or from context

One question remains: if the response of the query with counterparametric context Q' is a certain answer $x_1, \ldots, x_n$, 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 $\text{PPL}\left(x_1, \ldots, x_n \mid Q'\right)$ and comparing it to the distribution of perplexities on queries with added counterparametric context P_2 and P_3. The probability of this value being on one distribution or another might give us a good idea of the source the model used for generating the answer.

Base Query Q

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

Query with Counterparametric Context Q'

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

Parametric Answer Tokens $p_1, \ldots, p_n$

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

Counterparametric Answer Tokens $q_1, \ldots, q_m$

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

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

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

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

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

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

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

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

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

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

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

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

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

$P_2 > P_3 \quad \Longrightarrow \quad$ **Contextual**

Figure 4: Example of perplexity calculation for the parametric and counterparametric answers in a query with the counterparametric context. For each large language model, we calculate the probability of getting each parametric token $p_1, \ldots, p_n$ and each counterparametric token $q_1 \ldots q_m$ in the query with added context Q', and we accumulate that into two perplexity values. Note that, due to teacher forcing, the calculation finds the probability of the next token p_i *given the previous tokens of the searched answer* $p_1, \ldots, p_{i-1}$ rather than given the most likely tokens. For example, once we feed the string "`National Gallery of Canada in`", the probability of the next token being "`Ottawa`" is very high. In the same way, despite the perplexity of the token "`collection`" following the initial token "`the`" is very high, the perplexity of the following tokens in the parametric answer is considerably lower.

4 Results

We followed the methods explained in Section 3 to create a new dataset and run the models listed Section 3.2.1 in order to measure their knowledge grounding of each one of these models when adding counterparametric context.

This section presents the results from these runs.

4.1 Creating a representative dataset of questions

As described in Section 1.4.1 and explained in Section 3.1, we require a new and diverse dataset in order to run this data and answer the research question.

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 their historical 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, following and enhancing the question-building approach used by Yu et al..

The final 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 after merging the base questions with the objects of each respective category.

4.2 Building an experimental framework to understand the source of an LLM's answer

4.2.1 Building and running the framework

The code used for running this framework is present in Appendix C. This code roughly follows the diagram on Figure 3 to run the following steps.

1. Generate questions from base questions and objects: `combine_questions`.
2. Gather the parametric answers for each model: `QuestionAnswerer.answerChunk`.
3. Shuffle them to create counterfactual answers: `sample_counterfactual_flips`.
4. Build new queries with these counterfactual answers as context: `Question.format`.
5. Run the models again, and gather the corresponding answer type for each answer from the results: `QuestionAnswerer.answerCounterfactuals`.

The models were run on a server with 48 Intel(R) Xeon(R) CPU 3GHz CPUs, 376GB of RAM, and 2 NVIDIA A100 GPUs with 80GB of VRAM each that was kindly provided to Artificial Intelligence MSc students in City, University of London.

We estimate that it's possible to run this framework for all but the largest model, `Meta-Llama-3.1-70B`, using a single A100 GPU.

Appendix B explains the many options that can be run on `knowledge_grounder.py` to re-run this experiment or run similar ones.

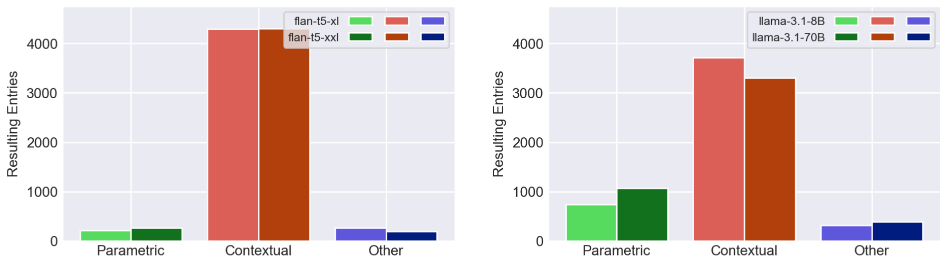
4.2.2 Framework Results

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
flan-t5-xl	248	4284	228
flan-t5-xxl	242	4304	214
Meta-Llama-3.1-8B-Instruct	745	3662	353
Meta-Llama-3.1-70B-Instruct	1070	3303	387

Table 8: Amount of answers of each category when running our dataset on each of the four models. Seq2Seq models tend to generate using the parametric context more often than Decoder-only models, while for the latter the size of the model makes a large difference.

.



(a) Seq2Seq models (b) Decoder-only models

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

As hypothesised in Section 1.4.2, there are vast differences on how the models of different types and sizes act when presented with a context that contradicts their knowledge. This section contains an overview of the differences, which investigated further in Section 5.

A similar pattern emerges in most (but not all) of the categories, which can be seen in Tables 10 and 11 and Figures 6 and 7.

4.2.3 Calculating the attention of the context and question of each query

Using the method described in Section 3.2.3, we can find the attention each model gives, on average, to the tokens in the context part of the query, and how much to the rest.

Query Part	flan-t5-xl	flan-t5-xxl	Meta-Llama-3.1 -8B-Instruct	Meta-Llama-3.1 -70B-Instruct
Context	0.18	0.22	0.08	0.06
Rest	0.03	0.05	0.03	0.01

Table 9: Average normalised attention of the query on the tokens corresponding to the context of the query and o the rest. All models pay more attention to the context section of the query than to the rest, but Seq2Seq models tend to have a much higher ratio than 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. There is not a significant difference between categories.

	Meta-Llama-3.1-8B-Instruct			Meta-Llama-3.1-70B-Instruct		
	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 11: Results for running each one of the 10 categories separately on the Decoder-only models, there are a few differences in the results for different categories, which is likely caused by the average answer size.

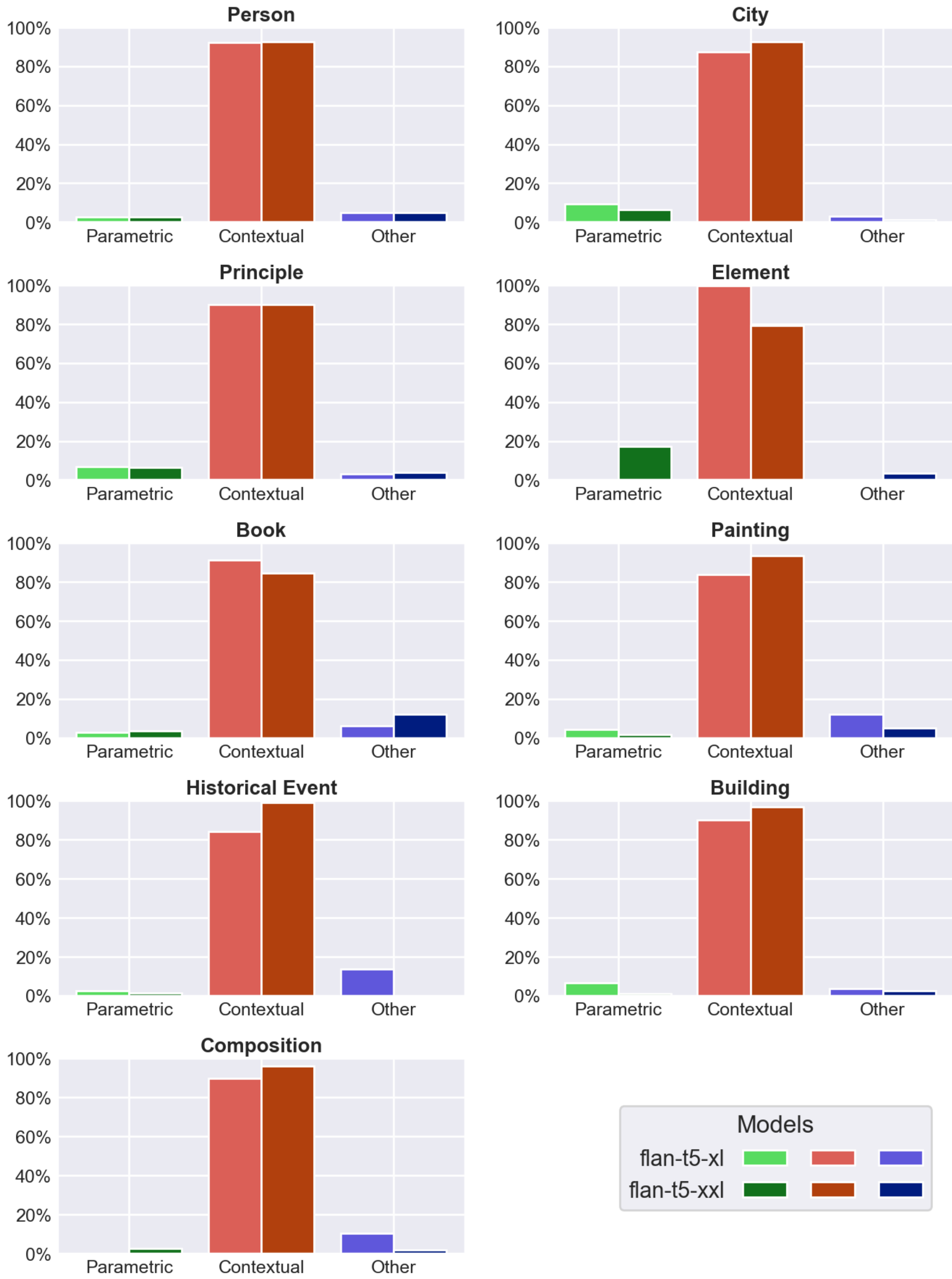


Figure 6: Results of running Seq2Seq models on the queried data, grouped by category. This plots the information shown in Table 10, and we can confirm that there isn't a significant difference in the source of the knowledge used to generate the answer depending on category.

Figure 7: Results of running decoder-only models on the queried data, grouped by category. This plots the information shown in Table 11. Contrasting this to the previous plot, we can see that there is a significant difference in contexts that tend to have questions resulting in shorter answers, such as books and historical events, compared to sections that tend to have longer answers, such as persons.

4.3 Enhancing the framework to understand the reasoning behind each answer

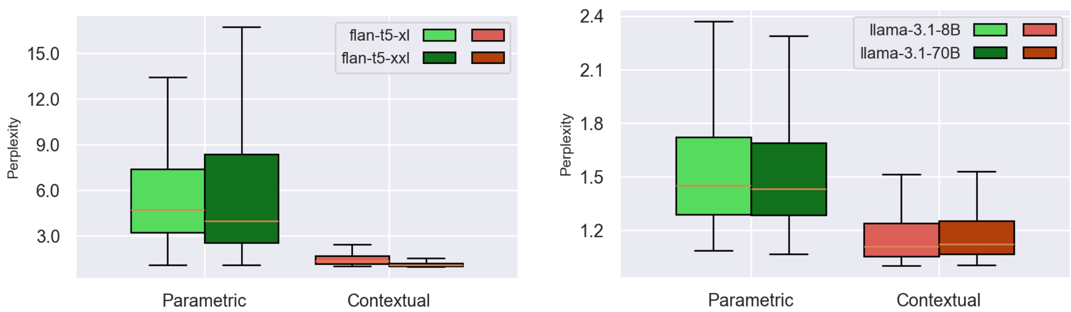
We calculate the resulting perplexity of each query as explained in Section 3.3. These are accumulated in three distributions, depending on answer type, which are summarised in Tables 12 and 13 and Figure 8, and grouped by different categories in Figures 9 and 10, and later grouped by category in Figures 9 and 10.

	flan-t5-xl		flan-t5-xxl	
	Parametric	**Contextual**	**Parametric**	**Contextual**
count	588	4172	491	4269
mean	7.02	1.55	12.01	1.24
std	11.25	0.56	18.91	0.68
10%	2.60	1.08	1.47	1.00
25%	3.21	1.18	2.54	1.02
50%	4.71	1.37	4.00	1.08
75%	7.40	1.69	8.37	1.22
90%	10.72	2.32	44.25	1.54

Table 12: Distribution of perplexity values for Seq2Seq models. The perplexity of **Parametric** answers is consistently higher than the one of **Contextual** answers.

	Meta-Llama-3.1-8B-Instruct		Meta-Llama-3.1-70B-Instruct	
	Parametric	**Contextual**	**Parametric**	**Contextual**
count	289	4471	383	4377
mean	1.64	1.20	1.56	1.22
std	0.87	0.30	0.46	0.31
10%	1.18	1.03	1.20	1.03
25%	1.28	1.05	1.28	1.06
50%	1.45	1.11	1.43	1.12
75%	1.72	1.23	1.68	1.25
90%	2.22	1.43	2.04	1.49

Table 13: Distribution of perplexity values for Decoder-only models. The same differences in Table 12 appear here, but to a lesser degree.



(a) Seq2Seq models

(b) Decoder-only models

Figure 8: Perplexity distribution according to model architecture and size. These represent the same distributions as Tables 12 and 13.

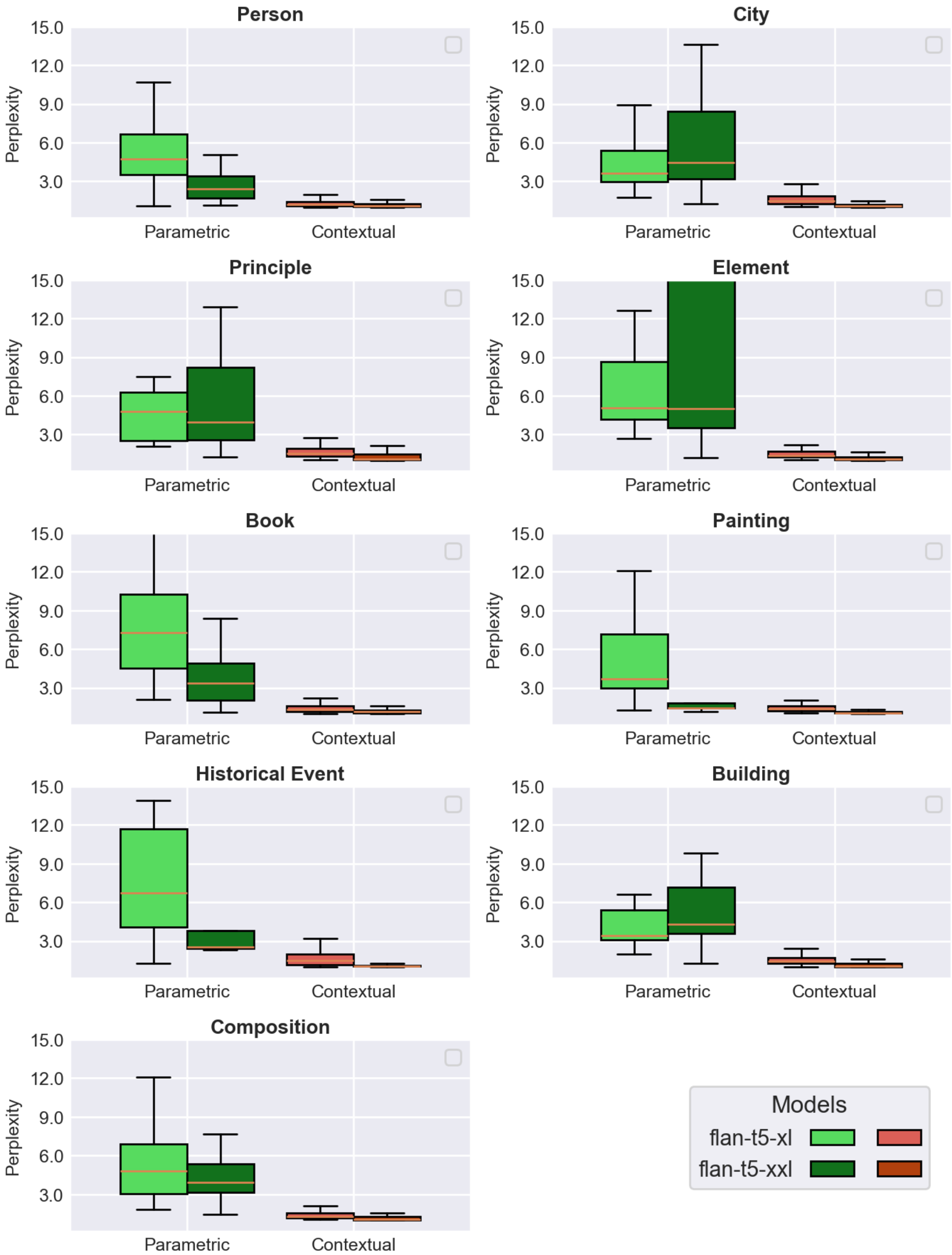


Figure 9: Box plots representing the distribution of the perplexities when running both Flan-T5 models, grouped by category. There are considerable differences in the distribution of perplexity values for **Parametric** answers, but not so much for **Contextual** answers. Still, the differences between this two are consistently large.

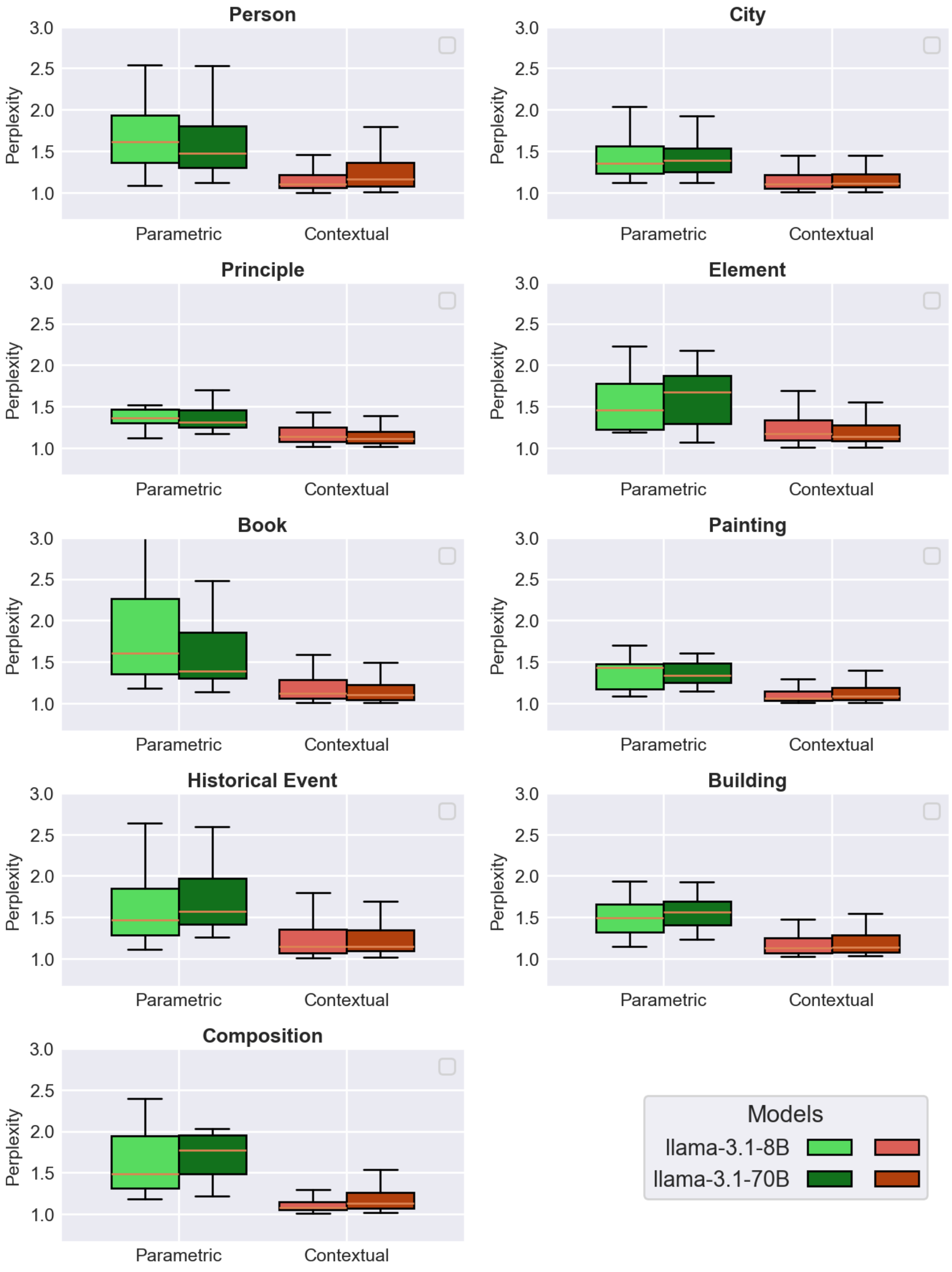


Figure 10: Box plots representing the distribution of the perplexities when running both Llama models, grouped by category. The entire distribution of perplexities is much smaller, and the differences in these for **Parametric** answers are less variable as in the previous plots for Seq2Seq models.

5 Discussion

Section 4 presented results from generating the question dataset and running the framework to understand the role of knowledge grounding in a variety of models and their parametric knowledge in question-answering. This section explains these results, and discusses what they mean for our research question.

5.1 Model architecture and memorised knowledge

Section 4.2.2 presents the results of running our framework on the provided questions on two different model architectures: Seq2Seq models and Decoder-only models. Tables 10 and 11 show these results split by category of question.

The results are clear: **Seq2Seq models tend to answer questions from their contextual knowledge rather than from their inherent knowledge more often than Decoder-only models**. These results persist across different question categories and are consistent regardless of answer types and lengths

These results demonstrate that, in the framework of question-answering when using RAG to fetch contextual data from an index, Seq2Seq models will tend to have fewer hallucinations that contradict this index than Decoder-only models.

We propose two hypotheses that could explain these differences.

5.1.1 Advantages of the Encoder-Decoder Architecture

As described in Section 2.2, Seq2Seq models such as `Flan-T5` are encoder-decoder models that process the entire context of the query in the encoder component before passing it to the decoder, which could increase the weight given to the context itself.

We can test this hypothesis by finding the mean attention of the tokens corresponding to the context and to the rest of the query, and using the method discussed in Section 3.2.3.

The results presented in Section 4.2.3 are consistent with this theory: Seq2Seq models allocate more attention to the contextual tokens of the query than Decoder-only models.

5.1.2 Different training data and fine-tuning

It's possible that these result doesn't come from the model architecture, but from the bias caused by their training methodology.

The `Flan-T5` models were trained on masked token generation and later fine-tuned on question-answering about passages (Chung et al. 2022). This requires strong alignment between query and answer, which encourages the model to focus on the input context and makes it more likely to take the answer from the RAG-provided context.

Llama models were trained mainly on open-ended text generation, which relies more on parametric data.

It's possible that the deficiencies of knowledge grounding in Llama models might come simply to not being trained on related tasks.

5.2 Model size and memorised knowledge

Section 4.2.2 also shows differences in how models of different sizes process information in queries with counterparametric context.

5.2.1 Seq2Seq Models

While the average results are very similar, which is likely due to the properties of Seq2Seq models discussed in Section 5.1, the models seem to be significantly different for queries of category `element`, `historical_events`, and a few others.

Unfortunately, it's possible that the significance of these results comes down to failures on the large language models: for reasons not we do not fully understand yet, `flan-t5-xl` often produces nonsensical answers to questions requiring short responses, such as elements' names in the periodic table on historical dates. This behaviour is different than `flan-t5-xxl`, which generally provides correct answers in these cases.

When removing the categories with a significant amount of short answers and their nonsensical answers by the smaller model, the results of both Seq2Seq models seem similar. Therefore, we can conjecture that **the size of a Seq2Seq model does not significantly affect its knowledge grounding**.

5.2.2 Decoder-only Models

Section 4.2.2 shows a very different result for Decoder-only models. Surprisingly, the smaller model `Meta-Llama-3.1-8B-Instruct` has *better* knowledge grounding than the larger model `Meta-Llama-3.1-70B-Instruct`.

We already established that decoder-only models rely on parametric knowledge to a greater degree than Seq2Seq models. Larger models have a vast internalised knowledge base accumulated from expensive training data, which can lead to increased confidence in their parametric knowledge.

It's possible that larger Decoder-only models are able to use their parametric knowledge to interpret the answer to the question in more ways that contradict the contextual knowledge. The extra information encoded on the model's weights can produce more varied evidence against the contextual answer.

With this information, we can say that **the size of Decoder-only models *does* affect its knowledge grounding, and when enhancing queries with RAG it might be preferable to use a smaller model**.

This is consistent with similar results found for other Decoder-only models, such as Pythia and GPT-2 (Yu et al. 2023).

5.3 What are all these Others?

Section 4.2 showed that a significant minority of responses to queries with counterfactual context are **Other**: answers that aren't equal to either the parametric nor the contextual data. The numbers of these entries, per model, are presented in Table 14.

	Flan-T5-XL	Flan-T5-XXL	Meta-Llama-3.1 -8B-Instruct	Meta-Llama-3.1 -70B-Instruct
Other	260	192	312	387

Table 14: Amount of **Other** entries, that is, results where the answer was not either the parametric or contextual answer.

By manually checking these results, we can understand the reason why the model chose these answers comes down to one of the following seven cases.

1. **Different phrasing of a parametric answer**
   There are many answers where the model provides the parametric answer phrased with the format of the counterparametric context given in the query.
2. **Plain incorrect answers**
   Sometimes, adding counterfactual context to the query causes the model to produce an incorrect answer, which is different the answers from both the parametric knowledge of the model and the given context.
3. **Question misinterpretation due to the context**
   Some questions can be ambiguous or have a low probability of another answer. By adding a context with a counterfactual answer, the model can misinterpret the question and answer that's correct different to both the context and the parametric answer.
4. **Negating the context**
   This is an interesting one: if the model has an answer in its parametric knowledge that contradicts the data on its context, then it interpret the context as part of the question and adds its negation as part of the answer.
5. **Different phrasing of the context**
   Much less common than point 1, models sometimes give the same answer as provided in the query's context but in the format of the parametric answer.
6. **Correct answer, just different than the parametric answer**
   Some questions have multiple correct answers, and adding counterfactual context can cause the model simply choose different one from its parametric memory.
7. **Mixing elements of both parametric answer and context**
   The final answer contains elements of the parametric answer combined with elements of the given. This produces an answer that's different to both the parametric and contextual answer, but with parts of both of them. The cause of this is likely the greedy decoding used to find the answers, as explained in Section 3.2.2.

Examples of each one of these types can be found in Table 16.

Does the architecture and size of the model affect the distribution of each type of **Other** answer? Table 15 contains the amount of answers for each model.

Type	Flan-T5-XL	Flan-T5-XXL	Meta-Llama-3.1 -8B-Instruct	Meta-Llama-3.1 -70B-Instruct
(**Parametric**)	248	242	745	1070
(**Contextual**)	4284	4304	3662	3303
1.	0	0	116	234
2.	6	3	50	15
3.	0	0	13	8
4.	0	0	20	61
5.	241	170	33	38
6.	7	16	63	23
7.	6	3	17	8

Table 15: Different types of **Other** answers per model (with amount of **Parametric** and **Contextual** added for comparison). The two most notable groups are **1.**, which contains parametric answers with different phrasing, and **5.**, which contains counterfactual answers with different phrasing.

Surprisingly, there is a large difference in the distribution of answers that don't come either from the model or from the given context.

In the case of Seq2Seq models, the majority of **Other** answers are **Contextual** answers with a different format due. This is consistent with the previous result, where the vast majority of their answers came from the query's context.

The majority of **Other** answers in Decoder-only models are the opposite: **Parametric** answers expressed in the format of the context. However, the reasons are much more varied and this would be an interesting topic of discussion in future research.

Part of the reason for many of these answers are not equal to the parametric answer coming from the model's inherent knowledge or the answer given in the query's context, particularly on the Seq2Seq models, is due to the strict comparison function we use to define answer type. This is an area that should be improved; Section 6.3.1 gives more information and various suggestions on how to compare results that might be relevant on future work.

5.4 Differences in distribution of perplexity scores

The distribution of perplexity scores, shown in Section 4.3, shows a significant difference in the perplexity scores between **Parametric** and **Contextual** answers for all four models. This is even more marked on Seq2Seq models, where for both models the 90% percentile of the **Contextual** answer is lower than the 10% percentile of the **Parametric**.

In fact, this confirms our conjecture that **models tend to be surprised when finding an answer that contradicts the context of the query**. This conjecture seems to be true regardless of model size or question type (see Figures 9 and 10).

Reason	Question	Parametric	Counterfactual	Contextual
1.	Who was the primary leader associated with The Reforms of Diocletian?	Diocletian Himself	Caracalla, a Roman Emperor	Diocletian, a Roman Emperor
	In which city is Louvre Pyramid located?	Paris, France	the city of Valladolid, in the state of Yucatan, Mexico	the city of Paris, in the country of France
2.	In which period of the periodic table is Silver located?	5 of the periodic table	3 of the periodic table	4 of the periodic table
3.	What was the duration of Queen Elizabeth II's Platinum Jubilee?	12 months	approximately 100 years	approximately 70 years
	What is the nearest major body of water to Cusco?	Lake Titicaca	the North Sea	the Pacific Ocean
4.	What is the name of the main protagonist in One Flew Over the Cuckoo's Nest?	Randle McMurphy	Achilles	Not Achilles
	What is Frida Kahlo primarily known for?	her self-portraits and her depiction of Mexican culture	his theories on communism and his critique of capitalism	her artwork and her life story, not for his theories on communism or critique of capitalism
5.	How many pages are in One Flew Over the Cuckoo's Nest?	320 pages	480 pages in a standard edition	480 pages
6.	Who is credited with the discovery of Conservation of Energy?	Julius Robert Mayer	Alfred Wegener	James Joule[1]
	What is the name of the main protagonist in The Great Gatsby?	Nick Carraway	Liesel Meminger	Jay Gatsby[2]
	What educational institution did Srinivasa Ramanujan attend?	The University of Madras	The University of Vienna	the University of Cambridge[3]
7.	What is the date of death of Vladimir Lenin?	January 21, 1924	March 28, 1941	March 21, 1924
	What's the main nationality of Mozart?	Austria	English-Born American	American-born English-born Austrian

[1] Discovery of the conservation of energy is credited to both Julius Robert Mayer and James Joule.
[2] Nick Carraway and Jay Gatsby are co-protagonists in The Great Gatsby.
[3] Srinivasa Ramanujan attended both the University of Madras and the University of Cambridge.

Table 16: Examples of different types of **Other** answers when running the `Meta-Llama-3.1-8B-Instruct` model, with the categories listed and explained in Section 5.3.

The cause for the difference in the perplexity values for different architectures is likely the same as discussed in Section 5.1 on why the `Flan-T5` models generally have better performance on these tasks than `Llama` models.

Model Architecture By processing the entire input in one go, the Seq2Seq encoding step processes information in the context before starting decoding. In Decoder-only models, the shift from going from contextual to parametric knowledge is less abrupt, resulting in a less pronounced change in perplexity.

Training Data `Flan-T5` models are trained in data where getting information from the query itself is more relevant. In a sense, they are biased to being more surprised when producing a parametric answer.

5.4.1 Can we use the perplexity to predict the source of an answer?

Seeing the current results, it's tempting to attempt to create an estimator which, given information about the perplexity of an answer alone and without making any additional query, can estimate whether that answer in a query with added context came from the **Parametric** knowledge of the context or from the **Contextual** information in the query.

This can be useful in practice: if a RAG-enhanced model finds an answer with high perplexity, it can attempt to re-run the RAG indexing to find a more relevant answer and possible prevent a hallucination.

The results in Tables 12 and 13 seem consistent with Figure 11 in that such predictor would work better in Seq2Seq models, particularly on `flan-t5-xxl`: by re-running the RAG indexer on answers with perplexity of over 1.5, we can prevent hallucinations on 90% of the answers that came from parametric knowledge while not reindexing on 80% of the total questions.

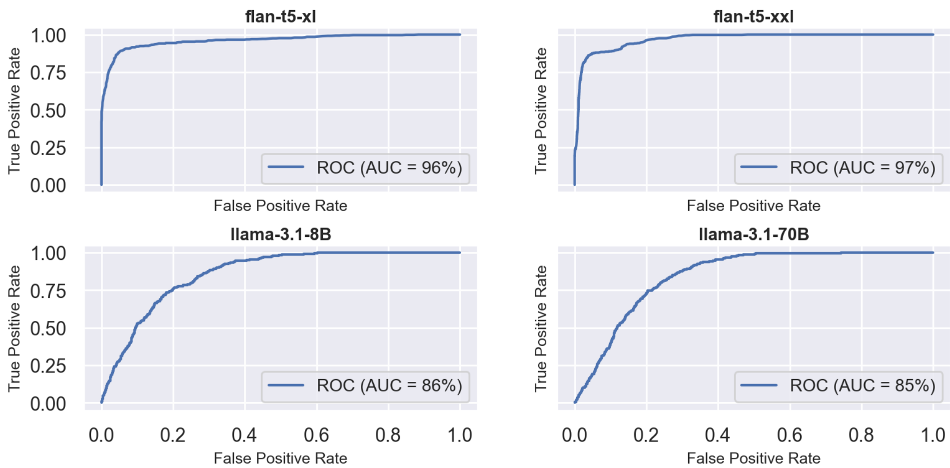


Figure 11: ROC curve of the four tested models to find out whether an answer came from the **Parametric** knowledge of a model using the perplexity of the answer.

6 Evaluations, Reflections, and Conclusions

6.1 Overview of the Research Question

Section 1.2 introduced a simple research question: **how does a large language model respond when given information that contradicts its inherent knowledge, and why?**

Using the methods outlined in Section 3, and analysing the results outlined in Section 4 and discussed in depth in Section 5, we can reach the following conclusions.

Seq2Seq models provide better results in answering data from contextual knowledge than Decoder-only models.
As described in Section 5.1, Seq2Seq models have an inherent advantage when gathering data from the context of the query when this contradicts its inherent parametric knowledge. It's possible that this difference comes down to training data rather than model architecture alone.

Size does not have a large impact in Seq2Seq models, but larger models are at a disadvantage in Decoder-only models.
As described in Section 5.2, the architecture of decoder-only models that biases them towards answering questions from their parametric knowledge more often also amplifies this effect in larger models.

Categorising answers is hard, and better methods are needed.
A lot of the answers we got in these experiments seemed to come from neither the query's context nor the parametric memory of the models. In Section 5.3 we demonstrate that the majority of those answers *do* come from those models, but that our methods to compare the answers are not good enough to categorise them properly.

Large language models tend to be surprised when finding an answer that contradicts the context of the query.
As seen in Section 5.4, models that choose a **Parametric** answers have higher perplexity than models that choose **Contextual** answers. This is consistent across all question categories and model architectures and sizes. This result is more marked on Seq2Seq models, likely for the same reasons why those models tend to generate **Contextual** answers more often.

It's possible to create an estimator to know whether an answer came from the parametric memory of a model or from the provided context.
Section 5.4.1 explains how to build such an estimator. This can be useful in cases where accuracy is crucial, but reindexing to get new context is expensive.

6.2 Personal Reflections

Working on this thesis has been a very positive and enriching experience.

Starting the project from an unpublished draft (Whitehouse et al. 2023) helped me ground my personal knowledge of the current state of the research on retrieval-augmented generation and analysis of large language models. I am grateful to Dr. Whitehouse, the main author of the draft, who has been of great help in giving me ideas and direction when working on this research.

The topic of the thesis changed early on from understanding knowledge grounding on the conflation of retriever and generator on retrieval-augmented models to understanding it for general large language models with added contextual information. This was, in part, caused by the cutting-edge nature of this research area: over half of the papers cited were published after the year 2020, and two of them were published earlier in the year 2024.

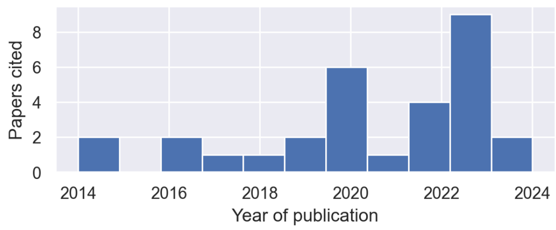


Figure 12: Year of publication of papers cited in this thesis. These topics are novel!

One of the main strengths of my work in this thesis was having built a solid experimentation infrastructure early on, which allowed me to iterate quickly on experiments and ideas. The program to build, run, and analyse the counterparametric-enhanced queries (which is present in Appendix C and in this project's repository) is one of the strong points of this project, and I'm sure it will be a valuable tool in future research projects.

Despite settling on the final thesis idea early, this thesis suffered from lacking a clear and well-defined research question, which emerged late in the process. I consider this delay to have been one of the main sources of inefficiencies, since it lead to executing long experiments that contributed little to the final thesis. One of my main takeaways from this research is the necessity to align on a well-defined research question early.

This thesis taught me how to adapt to the rapidly-changing research landscape of sequence analysis and artificial intelligence. A lot of the methods used in this work are taken directly from previous academic papers, and this allowed me to test my hypothesis quickly while keeping a deep understanding of the context I am working on.

Most of all, I'm happy to have learned a lot about the areas of large language models and to have been able to collaborate a bit in this very new area of research. I plan to continue this work in the future, and I'm looking forward to contributing to the research on this area!

6.3 Future Work

6.3.1 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 17 provides some examples of answers where the answer should have been categorised as **Parametric** or **Contextual**.

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 17: 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 a **Parametric** or **Contextual** answer with a slight formatting difference.

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

6.3.2 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, and we can experiment with an analysis similar to the one done in this thesis for those models.

6.3.3 Fine-tuning a LLM for a RAG Context

In this research we proved that certain models architectures are preferable to others for answering from data from the context when it contradicts the parametric knowledge of a model.

Existing retrieval-augmented language models, such as Atlas and Retro, are trained on existing models along with an index. We can continue this approach by fine-tuning an existing LLM, such as the `Flan-T5` or Llama models used in this thesis, specifically to be able to answer these kinds of queries better.

Glossary

This section presents a small glossary with explanations of some of the terms used in this thesis.

Category
: One of the many topics for each one of the question, which are listed in Table 7.

Base Questions
: A question that refers to a category, but does not refer to any particular object.

Objects
: An object of a certain category that can be added to a base question. The final queries are the cross product of all base questions and objects for each category; see Table 1 for an explanation.

Parametric Answers
: The answers given by the model for a particular query when not adding any context. These answers come solely from the parametric, learned knowledge of the model.

Counterfactual Answer
: A randomly-sampled parametric answer from another question, which is guaranteed to be different from this counterfactual answer. Table 4 contains an example of these.

Parametric, Contextual, and Other answers
: The answers given by the model for a particular query when adding counterfactual context, named like that depending on whether they were taken from the model's parametric memory, from the context of the query, or from somewhere else. Section 3.2.2 gives an explanation of how these are generated and categorised.

Perplexity of an answer
: The perplexity is a metric in information theory that measures how "surprised" a model is at finding a particular answer. This is explained in Section 3.3, where these answers are also categorised as **Parametric** or **Contextual** depending on which one is the most surprising.

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`

Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., de Las Casas, D., Hendricks, L. A., Welbl, J., Clark, A., Hennigan, T., Noland, E., Millican, K., van den Driessche, G., Damoc, B., Guy, A., Osindero, S., Simonyan, K., Elsen, E., Rae, J. W., Vinyals, O. & Sifre, L. (2022), 'Training compute-optimal large language models'.
URL: *https://arxiv.org/abs/2203.15556*

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., K"uttler, H., Lewis, M., Yih, W.-t., Rockt"aschel, T., Riedel, S. & Kiela, D. (2020), 'Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks', *Advances in Neural Information Processing Systems* **33**, 9459–9474.

Longpre, S., Hou, L., Vu, T., Webson, A., Chung, H. W., Tay, Y., Zhou, D., Le, Q. V., Zoph, B., Wei, J. & Roberts, A. (2023), 'The flan collection: Designing data and methods for effective instruction tuning'.
URL: *https://arxiv.org/abs/2301.13688*

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`

Martin, L., Touvron, H., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., Bikel, D., Blecher, L., Ferrer, C. C., Chen, M., Cucurull, G., Esiobu, D., Fernandes, J., Fu, J., Fu, W., Fuller, B., Gao, C., Goswami, V., Goyal, N., Hartshorn, A., Hosseini, S., Hou, R., Inan, H., Kardas, M., Kerkez, V., Khabsa, M., Kloumann, I., Korenev, A., Koura, P. S., Lachaux, M.-A., Lavril, T., Lee, J., Liskovich, D., Lu, Y., Mao, Y., Martinet, X., Mihaylov, T., Mishra, P., Molybog, I., Nie, Y., Poulton, A., Reizenstein, J., Rungta, R., Saladi, K., Schelten, A., Silva, R., Smith, E. M., Subramanian, R., Tan, X. E., Tang, B., Taylor, R., Williams, A., Kuan, J. X., Xu, P., Yan, Z., Zarov, I., Zhang, Y., Fan, A., Kambadur, M., Narang, S., Rodriguez, A., Stojnic, R., Edunov, S. & Scialom, T. (2023), 'Llama 2: Open Foundation and Fine-Tuned Chat Models'.
URL: *https://arxiv.org/abs/2307.09288*

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`

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., Rodriguez, A., Joulin, A., Grave, E. & Lample, G. (2023), 'Llama: Open and efficient foundation language models'.
URL: *https://arxiv.org/abs/2302.13971*

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
SchrÃ¶dinger Equation,principle
Special Relativity,principle
Aluminum,element
Barium,element
Bismuth,element
Bromine,element
Calcium,element
Carbon,element
Chlorine,element
Chromium,element
Copper,element
Gold,element
Helium,element
Hydrogen,element
Iodine,element
Iron,element
Lead,element
Lithium,element
Magnesium,element
Manganese,element
Mercury,element
Neon,element
Nitrogen,element
Oxygen,element
Phosphorus,element
Plutonium,element
Potassium,element
Radon,element
Silicon,element
Silver,element
Sodium,element
Sulfur,element
Thorium,element
Tin,element
Titanium,element
Uranium,element
Zinc,element
Argon,element
Boron,element
Cobalt,element
Fluorine,element
Gallium,element
Krypton,element
```

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

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

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

Listing 3: All objects which will be conbined with the questions in Listing 1.

B Grounder Usage and Documentation

This appendix provides a brief overview of how to use the program to run the analyses in this thesis.

The entire approach is done in Python, and can be run from the single file `knowledge_grounder.py`. The code of the program is provided in Appendix C, and also separately in the main repository for this thesis, the dedicated code repository in `https://github.com/mfixman/knowledge-grounder`, and attached to the submission area.

The code should be run concurrently with the source data present in Appendix A and in the two repositories. The result, if run with the `--output-dir` option, is one CSV file per model with information about its knowledge grounding.

B.1 Code description and recommendations

The code downloads and uses large language models from the Huggingface dataset. Many of the models are large, so it might be useful to download them using the Huggingface CLI first as detailed in Appendix B.1.

```
$ huggingface-cli download --repo-type model 'meta-llama/Meta-Llama-3.1-70B'
```

B.2 Code usage

The code usage is explained well when running the program with the `--help` argument.

```
$ python knowledge_grounder.py --help
usage: knowledge_grounder.py [-h] [--debug] [--lim-questions LIM_QUESTIONS]
    [--device {cpu,cuda}] [--models model [model ...]] [--offline] [--rand]
                         [--max-batch-size MAX_BATCH_SIZE] [--per-model]
    [--output-dir OUTPUT_DIR] [--runs-per-question RUNS_PER_QUESTION]
                base_questions_file objects_file

  base_questions_file   File with questions
  objects_file          File with objects to combine

  -h, --help            show this help message and exit
  --debug               Go to IPDB console on exception rather than exiting.
  --lim-questions LIM_QUESTIONS Question limit
  --device {cpu,cuda}   Inference device
  --models model [model ...] Which model or models to use for getting parametric
    data
  --offline             Run offline: use model cache rather than downloading new
    models.
  --rand                Seed randomly rather thn using the same seed for every
    model.
  --max-batch-size MAX_BATCH_SIZE
                        Maximum size of batches.
  --per-model           Write one CSV per model in stdout.
  --output-dir OUTPUT_DIR
                        Return one CSV per model, and save them to this directory.
  --runs-per-question RUNS_PER_QUESTION
                        How many runs (with random counterfactuals) to do for
    each question.
```

B.3 Example usage

```
$ python knowledge_grounder.py \
    --device cuda \ # Use CUDA (it's possible to use CPU for small models)
    --models llama flan-t5-xl flan-t5-xxl \ # List of models to try
    --output-dir outputs/ \ # Write outputs to this directory
    --rand \ # Randomly seed after every model. This will cause answers to vary
    from other runs.
    -- \
    data/base_questions.txt # File with {}-format base questions.
    data/objects.csv # File with objects.
```

Listing 4: Example usage: run three models with random data.

```
$ python knowledge_grounder.py \
    --device cuda \ # Use CUDA (it's possible to use CPU for small models)
    --models llama-70b \ # This is a large model; let's run it separately.
    --max-batch-size 70 \ # Smaller batch size to ensure the program won't run
    out of VRAM.
    --output-dir outputs/ \ # Write outputs to this directory
    --offline \ # Run offline; this will fail if the model is not previously
    downloaded.
    -- \
    data/base_questions.txt # File with {}-format base questions.
    data/objects.csv # File with objects.
```

Listing 5: Example usage: run one of the models with 0 seed, to ensure repeteability

C 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`*.

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

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

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

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

    parser.add_argument('--debug', action = 'store_true', help = 'Go to IPDB console on exception rather than exiting.')
    parser.add_argument('--lim-questions', type = int, help = 'Question limit')
    parser.add_argument('--device', choices = ['cpu', 'cuda'], default = 'cuda', help = 'Inference device')
    parser.add_argument('--models', type = str.lower, default = [], choices = Model_dict.keys(), nargs = '+', metavar = 'model', help = 'Which model or models to use for getting parametric data')
    parser.add_argument('--offline', action = 'store_true', help = 'Run offline: use model cache rather than downloading new models.')
    parser.add_argument('--rand', action = 'store_true', help = 'Seed randomly rather thn using the same seed for every model.')
    parser.add_argument('--max-batch-size', type = int, default = 120, help = 'Maximum size of batches. All batches contain exactly the same question.')

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

    parser.add_argument('--runs-per-question', type = int, default = 1, help = 'How many runs (with random counterfactuals) to do for each question.')

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

    args = parser.parse_args()

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

```

    del args.base_questions_file
    del args.objects_file

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

    return args

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

    # We want to set this environment variable to ensure that Huggingface
    # does not download models.
    if args.offline:
        os.environ['TRANSFORMERS_OFFLINE'] = '1'
    else:
        wandb.init(project = 'knowledge-grounder', config = args)

    # Combining the base questions and objects into the final questions.
    logging.info('Getting questions')
    questions = combine_questions(args.base_questions, args.objects, args.lim_questions)

    # Create the output dir, if necessary.
    if args.output_dir:
        try:
            os.mkdir(args.output_dir)
        except FileExistsError:
            pass

    logging.info(f'About to answer {len(questions) * len(args.models) * args.runs_per_question * 2} questions in total.')
    answers = {}
    for model in args.models:
        # Let's use the same seed for all models.
        # This ensues that results are reproducible.
        if not args.rand:
            random.seed(0)

        # Create a QuestionAnswerer object to answer these queries.
        # These objects are large, so ensure it's deleted before loading the next one.
        qa = QuestionAnswerer(
            model,
            device = args.device,
            max_length = 20,
            max_batch_size = args.max_batch_size,
            runs_per_question = args.runs_per_question,
        )
        model_answers = qa.answerQueries(questions)
        del qa

        if args.output_dir:
            count = lambda s: sum([x == s for x in model_answers['comparison']])
```

```python
            logging.info(f"\t{count('Parametric')} parametrics, {count('Contextual')} contextual, {count('Other')} others")

            # Write the information to a corresponding file for each model.
            model_filename = os.path.join(args.output_dir, model + '.csv')
            with open(model_filename, 'w') as out:
                print_parametric_csv(out, model_answers)

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

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

if __name__ == '__main__':
    args = parse_args()

    # If run with --debug, launch an IPDB console on exception.
    if not args.debug:
        main(args)
    else:
        with ipdb.launch_ipdb_on_exception():
            main(args)
```

Listing 6: `knowledge_grounder.py` is the main entry point and contains mostly argument parsing and output printing.

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

import logging
import math
import torch
import typing

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

from collections import defaultdict
from transformers import BatchEncoding

import ipdb

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

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

```python
    runs_per_question: int
    llm: Model

    def __init__(
        self,
        model: Union[str, Model],
        device: str = 'cpu',
        max_length: Optional[int] = None,
        max_batch_size: Optional[int] = None,
        runs_per_question: Optional[int] = None,
    ):
        self.device = device
        self.max_length = max_length or 20
        self.max_batch_size = max_batch_size or 120
        self.runs_per_question = runs_per_question or 1

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

        model = typing.cast(Model, model)
        self.llm = model

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

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

        # Get the tokens of the question and generate the parametric answer.
        base_tokens = self.tokenise([q.format(prompt = self.llm.prompt) for q in questions])
        parametric = self.generate(base_tokens)

        parametric_output = self.decode(parametric)
        base_proba_output = self.perplexity(base_tokens, parametric)

```

```
        # We possibly want several runs here, each with a different randomly sampled set of counterfactuals.
        for run in range(self.runs_per_question):
            run_output: dict[str, list[Any]] = dict(
                parametric = parametric_output,
                base_proba = base_proba_output,
            )

            # Sample the counterfactuals and add them to the output.
            run_output['question'] = questions
            flips = sample_counterfactual_flips(questions, run_output['parametric'])
            counterfactual = parametric[flips]

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

            # Answer the counterfactuals, and union this dictionary to the output dictionary.
            run_output |= self.answerCounterfactuals(questions, run_output['counterfactual'], parametric, counterfactual)

            # We want to compare each contextual answer to their parametric and counterfactual.
            run_output['comparison'] = [
                'Parametric' if self.streq(a, p) else
                'Contextual' if self.streq(a, c) else
                'Other'
                for p, c, a in zip(run_output['parametric'], run_output['counterfactual'], run_output['contextual'])
            ]

            # We also want to figure out if P_2 < P_3.
            run_output['preference'] = [
                'Parametric' if pp > cp else
                'Contextual'
                for pp, cp in zip(run_output['ctx_proba'], run_output['ctx_cf_proba'])
            ]

            for k, v in run_output.items():
                output[k].extend(v)

        return output

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

```python
        output: dict[str, Any] = {}
        ctx_tokens = self.tokenise([
            q.format(prompt = self.llm.cf_prompt, context = context)
            for q, context in zip(questions, counterfactuals)
        ])

        contextual = self.generate(ctx_tokens)

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

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

        output['context_attn'], output['question_attn'] = self.avgSelfAttentions(ctx_tokens)

        return output

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

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

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

            chunk_output = self.answerChunk(chunk)

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

        return dict(output)

    def fakeTokens(self) -> BatchEncoding:
        return self.tokenise(['[Context: Montevideo is located in Egypt] Q: What country is Montevideo located in? A: Montevideo is located in'])

    # Gets the scaled mean self-attentions of the context section of a query and
    # of the section after the context.
    @torch.no_grad()
    def avgSelfAttentions(self, queries: BatchEncoding) -> tuple[list[float], list[float]]:
        attentions = self.llm.attentions(queries)
        attention_mean = attentions.mean(dim = (1, 2))
        diag = attention_mean.diagonal(dim1 = 1, dim2 = 2)
        scaled = (diag - diag.min(dim = 1, keepdims = True)[0]) / (diag.max(dim = 1, keepdims = True)[0] - diag.min(dim = 1, keepdims = True)[0])

        context_left = ((queries.input_ids == self.llm.tokenizer.pad_token_id) | (queries.input_ids == self.llm.tokenizer.eos_token_id))
        context_right = ((queries.input_ids == self.llm.tokenizer.convert_tokens_to_ids(']')) | (queries.input_ids == self.llm.tokenizer.convert_tokens_to_ids('].')))

```

```
        context_area = ~context_left & (context_right.cumsum(dim = 1) == 0)
        later_area = context_right.cumsum(dim = 1) > 0

        context = scaled.clone()
        context[~context_area] = torch.nan

        later = scaled.clone()
        later[~later_area] = torch.nan

        return context.nanmean(dim = 1).cpu().tolist(), later.nanmean(dim = 1).cpu().tolist()

    # Tokenise a list of phrases.
    # [n] -> (n, w)
    def tokenise(self, phrases: list[str]) -> BatchEncoding:
        return self.llm.tokenizer(
            phrases,
            return_tensors = 'pt',
            return_attention_mask = True,
            padding = True,
        ).to(self.device)

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

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

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

        return sequences

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

```
        probs = self.batch_perplexity(query, self.batch_encode(answer))
        return probs.cpu().tolist()

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

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

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

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

Listing 7: `QuestionAnswerer.py` contains the `QuestionAnswerer` class which deals with the logic of answering parametric and counterfactual questions from a model

```
import logging

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

# Dictionary of models, containing all of the models aliases and their respective
    models.
Model_dict = {
    'llama': 'meta-llama/Meta-Llama-3.1-8B-Instruct',
    'llama-70b': 'meta-llama/Meta-Llama-3.1-70B-Instruct',
    'llama-405b': 'meta-llama/Meta-Llama-3.1-405B-Instruct',
    'flan-t5': 'google/flan-t5-base',
    'flan-t5-small': 'google/flan-t5-small',
    'flan-t5-base': 'google/flan-t5-base',
    'flan-t5-large': 'google/flan-t5-large',
    'flan-t5-xl': 'google/flan-t5-xl',
```

```
    'flan-t5-xxl': 'google/flan-t5-xxl',
    'gemma': 'google/gemma-2-9b-it',
    'gemma-27b': 'google/gemma-2-27b-it',
    'falcon2': 'tiiuae/falcon-11b',
    'falcon-180b': 'tiiuae/falcon-180b-chat',
    'falcon-40b': 'tiiuae/falcon-40b-instruct',
    'falcon-7b': 'tiiuae/falcon-7b-instruct',
    'distilbert': 'distilbert/distilbert-base-uncased-distilled-squad',
    'roberta': 'FacebookAI/roberta-base',
    'roberta-large': 'FacebookAI/roberta-large',
    'roberta-squad': 'deepset/roberta-base-squad2',
    'mixtral': 'mistralai/Mixtral-8x22B-Instruct-v0.1',
    'dummy': '',
}

# Virtual class containing a model.
# Derived classes should reimplement __init__ and logits, and attention.
# They should also save their tokeniser and HF model.
class Model(nn.Module):
    name: str
    model_name: str
    device: str

    tokenizer: AutoTokenizer
    model: AutoModelForCausalLM

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

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

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

        return DecoderOnlyModel(name, device)

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

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

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

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

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

```python

        kwargs = {}
        # Some models require special configuration.
        if 'llama' in name:
            kwargs = dict(
                pad_token = '<|reserved_special_token_0|>',
                padding_side = 'left',
            )
        elif 'gemma' in name:
            kwargs = dict(
                padding_side = 'right',
            )

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

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

    # Decoder-only generation with teacher forcing: calculate the next token given the previous forced tokens.
    @torch.no_grad()
    def logits(self, query: BatchEncoding, answer: BatchEncoding) -> FloatTensor:
        w0 = query.input_ids.shape[1]
        w1 = answer.input_ids.shape[1]

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

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

    @torch.no_grad()
    def attentions(self, queries: BatchEncoding) -> FloatTensor:
        outputs = self.model(**queries, output_attentions = True).attentions
        return torch.stack(outputs, dim = 1)

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

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

        kwargs = dict(
            padding_side = 'right',
        )
```

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

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

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

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

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

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

    @torch.no_grad()
    def attentions(self, queries: BatchEncoding) -> FloatTensor:
        outputs = self.model(
            **queries,
            output_attentions = True,
            decoder_input_ids = torch.full_like(queries.input_ids, self.tokenizer.pad_token_id).to(self.device),
        ).decoder_attentions
        return torch.stack(outputs, dim = 1)

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

        super().__init__(name, device)

    def __del__(self):
        logging.info(f'Deleting large model {self.name}')
```

```
        del self.model
        torch.cuda.empty_cache()

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

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

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

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

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

    def __getitem__(self, key):
        return self

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

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

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

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

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

Listing 8: `Models.py` contains the list of models and includes code that differentiates them.

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

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

```

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

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

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

        return False

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

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

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

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

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

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

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

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

        return formatted.strip()

# Returns the set product of a list of base question with the respective set of objects.
```

```python
def combine_questions(base_questions: list[str], objects: list[dict[str, str]], lim_questions: Optional[int] = None) -> list[Question]:
    questions = []
    for bq in base_questions:
        for obj in objects:
            q = Question.orNothing(obj = obj['object'], category = obj['category'], base_question = bq)
            if q is None:
                continue

            questions.append(q)

    if lim_questions is None:
        return questions

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

    return short_questions

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

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

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

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

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

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

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

        result[-1].extend(chunk)

    return result

# Prints a CSV file with the questions and resulting answers.
```

```python
def print_parametric_csv(out: typing.TextIO, answer: dict[str, list[Any]]):
    fieldnames = ['Num', 'Category', 'Base_Question', 'Object', 'Question', 'Prefix'] + list(answer.keys())

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

    for e, answers in enumerate(zip(*answer.values())):
        param = dict(zip(answer.keys(), answers))

        question = param['question']
        param.pop('question')

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

Listing 9: `Utils.py` contains various useful functions

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

import torch
from torch import tensor

from QuestionAnswerer import QuestionAnswerer

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

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

```
        ])

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

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

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

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

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

Listing 10: `test_QuestionAnswerer.py` is used to test some of the complicated bits of logic in QuestionAnswerer.