Using Retrieval Augmented Generation and Knowledge Graphs to Understand Climate Obstruction

Michael DeBellis  
N/A  
michaeldebellis.comSan Francisco, CA, USA  
https://orcid.org/0000-0002-8824-9577  
  
Jacob Gino  
Computer Science  
University of Wisconsin-MadisonMadison, WI, USA  
jacobkgino@gmail.com  
   
 Aadarsh Balaji  
Computer Science  
University of California, BerkeleyNew York, NY, USA  
aadarsh.balaji@gmail.com  
  
George Gino  
Computer Science  
Arizona State UniversityCupertino, CA, USA  
georgejgino@gmail.com

*Abstract*—Climate change is one of the most serious crises the human race has faced. Unlike previous crises such as the destruction of the ozone layer, the world has not come together to address the issue. At times this has been blamed on poor science communication. However, social scientists have realized that in reality the primary problem is that large corporations with vested interests in fossil fuels have orchestrated a campaign of disinformation and obfuscation which social scientists have labelled Climate Obstruction. This project is an attempt to collect the various resources (e.g., papers, databases, news articles) about Climate Obstruction into a knowledge graph using a Large Language Model (LLM) as the user interface via a Retrieval Augmented Generation architecture (RAG). The system provides a tool for researchers to have all the data in one location accessible via a natural language user interface. A long term goal is to use the rigor and logical foundation of the knowledge graph defined on a logical model using the Web Ontology Language (OWL) in combination with the LLM capability to model text meaning as vectors to rigorously define climate obstruction models that can be tested against data. I.e., to make the social science behind the analysis of Climate Obstruction a truly rigorous science. The current system is a proof of concept prototype. It provides useful functionality, but the team has not yet acquired the funding required to host the system on the Internet. The system runs as an Internet system (using localhost) already, all that is needed are some basic resources to support hosting the system.

Keywords — climate obstruction, Retrieval Augmented Generation (RAG), knowledge graph, Web Ontology Language (OWL), Large Language Model (LLM)

# Introduction

Climate change is one of the most serious crises the human race has ever faced. However, unlike previous crises such as the destruction of the ozone layer, the world has not come together to address the issue. At times this has been blamed on poor science communication [1] [2]. However, social scientists have realized that in reality the primary problem is that large corporations with vested interests in fossil fuels have orchestrated a campaign of disinformation and obfuscation which social scientists have labelled Climate Obstruction [3].

The goal of this research is to develop a Neurosymbolic (NS) model [4], [5] of the theory defined in [3]. A Neurosymbolic model combines two different techniques to represent meaning. The Web Ontology Language (OWL) utilizes symbolic representation and logic. LLMs model language using embeddings in a huge vector space created by the probabilistic analysis of terabytes of text.

There are several reasons such a model and knowledge graph can benefit researchers. In the short term, it serves as one central portal to find documents related to climate obstruction that can be queried and used to develop arguments, charts, etc. using a LLM. That is the primary goal of this project.

The longer term goal is to provide an example of a new way to approach the social sciences. We are creating another form of the model described in [3]. A model that should eventually be able to make falsifiable predictions. The Description Logic of the Web Ontology Language (OWL) provides a formal model of the Climate Obstruction model. Such a model has many benefits over a model only defined with words. An OWL model is a mathematical model just as any model in science. We can use the reasoner to prove that there are no logical errors in the model and to infer additional data based on the logical axioms.

In addition, the ability of an LLM to model meaning of text as vector space offers a completely different type of analysis using statistical probabilities which work better for much real natural language than any formal model.

As an example, the book defines a process model for the flow of various forms of influence. By creating a formal model we can analyze data and see if we can recognize the kind of patterns of influence flow.

Section II describes the methods and tools that we utilized. Section III describes our results to date. Section IV describes lessons learned from this work, and plans for the future.

References to names of OWL entities, Python functions and objects, and any other technical name are written in Courrier New 10 font. All of the code and the ontology are available via an open source license and can be found at our GitHub site: [6].

# Methodology

## Retrieval Augmented Generation (RAG)

Retrieval Augmented Generation (RAG) is an architecture that allows the system to utilize the semantic embedding and NLP understanding and generation of an LLM while using a curated corpus of documents as the knowledge-base rather than the default neural network of the LLM [7], [8]. The advantage of a RAG architecture is that for a specific domain, it addresses the two most significant issues with LLMs: black-box reasoning and hallucinations. A standard LLM does not know what it knows. It is not the case that understanding the reasoning of an LLM is simply difficult, it is as of 2025 an unsolved problem [9], [10]. This lack of explicit knowledge representation is the cause for both hallucinations and black-box reasoning. Black-box reasoning results because although an LLM can find sources to support its conclusion, those sources are post-hoc rationalizations. As demonstrated by [9], the specific cells and values that resulted in an LLM response are simply not accessible. This is also the cause for hallucinations. A standard LLM has no way to evaluate how strong a response is because it does not have access to the knowledge that was used to generate the response and hence has no way to evaluate whether that knowledge was a good match for the question. The RAG architecture solves both of these problems by substituting a curated corpus of documents for the domain knowledge of the LLM. Of course, as with any architectural decision there is a trade-off. An LLM has an incredible breadth of knowledge. A RAG system is much more narrow and typically focused on a fairly narrow domain such as customer support for a specific product or a specific sub-domain of medicine such as dental materials [11].

## Neurosymbolic Modeling

Neurosymbolic modeling refers to the integration of embedded vectors created by tools such as the Open AI API with knowledge graphs based on standards such as the Web Ontology Language (OWL) [4] [5]. Utilizing a knowledge graph rather than a relational database for the corpus provides extra capabilities such as explanations [12], inferencing , and graphical browsing [11].

In addition, the use of ontologies enables reuse of existing ontologies known as vocabularies that have commonly used models for specific problems. In this case, our ontology primarily is built on the Dublin Core vocabulary [13] for document meta-data, the Simple Knowledge Organization System (SKOS) [14] for modeling synonyms and meta-data related to creation and sources of ontology entities, the Gist Upper Model from Semantic Arts [15], the Universal Moral Grammar [16] for the definition of agents, causality, and moral responsibility, and the Cognitive Modules ontology to represent beliefs, facts, and intentions [17].

## Data Pipeline

A RAG system takes advantage of the architecture of LLMs that is defined by various services that can be used individually. E.g., the ability to define meaning for text as vectors in a large neural network vector space. The RAG architecture consists of two systems: the data pipeline and the run-time architecture. The data pipeline is displayed in Figure 1. It must be run before the system can be used.

The first step in the data pipeline is scraping web sites using Python libraries such as … to create CSV files. Rows in these files correspond to web sites and documents and the columns are headers such as abstract, introduction, conclusion, as well as domain specific headings such as jurisdiction for litigation databases and CAP code for databases about violations of advertising standards in the United Kingdom. In step two, we transform the CSV files in to knowledge graph objects. In this stage we do a direct transformation from column headings in the CSV files to data properties in the knowledge graph. This is similar to the Extract

Load Transform (ELT) model that distinguishes data lakes from Extract Transform Load (ETL) of data warehouses. The result is many new OWL instances with mostly string data properties.

In the third steps we create vectors for many of the string data properties. These vectors model the meaning of each string in a way that the LLM can understand so that the LLM can utilize these strings when creating answers to questions.

In step four we post process the objects created in step two. We turn certain strings into links to objects that we create or find. This process is known as transforming “strings to things” [13]. For example, we use the AllegroGraph Free Text Index (FTI) [14] to analyze the text strings that we generated vectors for and find references to entities such as organizations, places, events, etc. The FTI uses basic NLP tools such as stemming and bag of words to find matches from small phrases (one to ten words) and map them to the labels (which include skos:altLabel, skos:prefLabel, as well as rdfs:label) of entities in the knowledge graph. Such references create new has\_Topic property values. This property is an annotation property because text from the corpus can refer to classes (e.g., the Greenwashing class, which is a subclass of Communication and Event). Another example of this type of transformation is transforming the author strings from the meta-data for journal articles, web pages, etc. to objects. For this transformation we utilize the Nameparser python library [15]. We utilize a semicolon as the delimiter. Virtually all of the corpus documents that had authors used a semi-colon to delimit the names and the few that didn’t we first post processed so that they did. Nameparser allows us to parse through all the author names and separate them by first and last name. In addition, the HumanName Python class in Nameparser allows us to handle prefixes, initials, and hyphenated names. After the names are parsed and sorted, we search to see if an author with the same first and last name exists in the knowledge graph. If one does then we add an object property called has\_Author that points from the document to the author object. If one doesn’t exist it is created and the property value is asserted on it. We run the reasoner after transforming all the strings which adds additional data. E.g., in the case of authors it adds the inverse value is\_Author\_Of that points from the author to the document.

Once we have run the data pipeline we can use the RAG system. Although we will need to regularly re-run the pipeline to add new instances to the knowledge graph and vectors for new text strings.

## Run-time Architecture

Figure 2. shows the run-time architecture. The user enters questions via the streamlet user interface. Streamlit is a Python servlet user interface library. The user enters a question that is passed on to the Open AI API to create a vector for the question. That vector is matched to existing vectors in the system using a cosign nearest neighbor function. This is passed on to the Open AI API using the magic property llm:askMyDocuments. That finds the vectors in the Neurosymbolic knowledge base that are within the parameters specified as part of the SPARQL query. One parameter determines the maximum number of answers. This must be an integer. The default is five. The other is a floating point number that determines how close a match counts as a good match. This must be a floating point number between zero and one. The default is 0.7. Both of these parameters can be specified in the Streamlit user interface (see Figure 3).

When the user submits a new question, that string is passed to a function called do\_query. That function calls build\_query which has a SPARQL template that it fills in with the appropriate parameters based on the input from Streamlit. In this paper we will be demonstrating with the question: “What evidence exists that fossil fuel companies have used front groups or third parties to spread climate misinformation in Europe?”. The code excerpt below shows the SPARQL query[[1]](#footnote-1) that is generated from this input and with the parameters for the number of matching documents and required relevance.

SELECT \*

WHERE {bind ("What evidence exists that fossil fuel companies have used front groups or third parties to spread climate misinformation in Europe?" as ?query)

(?response ?score ?vec ?content) llm:askMyDocuments

(?query "climate\_obstruction" 5 0.7).

OPTIONAL{?doc :has\_Topic ?topic}

OPTIONAL {?super\_part :has\_direct\_part ?doc}}

This query is returned from build\_query to do\_query which then executes the query using the AllegroGraph Python client. The llm:askMyDocuments magic property does the following:

1. Utilizes the Open AI API to generate a vector for the user’s question.
2. Uses the cosine nearest neighbor function to find the nearest neighbor in the vector store for the repository that has a match value of (in this example) 0.7 or higher.
3. If multiple text strings are over the match threshold takes the N number of strings that are the best match (has the highest match number), where N is the parameter for maximum number of documents to utilize. In this example N = 5.
4. Sends the question, vectors, and N matching strings to the Open AI GPT-3.5 Turbo model that generates the response using the strings and vectors rather than its internal database. This is the *augmented* in RAG. We *augment* the prompt with data from our domain specific corpus rather than utilizing the much broader but shallower knowledge of the LLM.
5. Returns an answer that is bound to ?response in the llm:askMyDocuments magic property.

The rest of the query (some parts not shown for brevity) has various OPTIONAL statements to match properties such as the authors of a document, the part that a document is a sub-part of, and any entity identified by the FTI analysis as a topic of the document. These must be OPTIONAL because otherwise the SPARQL query would fail if any of the properties were missing and all properties are not always present for every document. The query is also passed to AllegroGraph’s Gruff graph visualization tool so that the knowledge graph objects associated with the specific question can be viewed graphically and can be further browsed by the user for additional information.

# Results and Discussion

The current prototype demonstrates:

1. How the system can be a useful research tool.
2. How defining a Neurosymbolic model can help formalize the theoretical concepts.

## A Retrieval Augmented Generation (RAG) Research Tool

Figure 3. shows an example of the user interface. The user can also click on the View answer graph in Gruff link and view the relevant objects for the query.

Figure 4 shows the objects for this specific query.

Figure 4 shows the objects for this specific query.

As illustrated in Figure 4, Gruff displays a color coded graph with a legend on the left that maps each node and link color to the datatype and property respectively. In most cases the datatype for a node is an OWL class and the property is an object property. The user can then further expand and change the layout of the graph. In this example, the user has expanded the super-classes of Falsehoods to see that it is a subclass of Green\_Washing which is ultimately a subclass of the Gist Event class. Figure 5 shows another query example, this time asking the question “What are examples of different kinds of Green Washing?” and Figure 6 shows the initial Gruff Graph and the graph after the user displays some subclass relations and uses

the “layout as tree” option. Space doesn’t permit a full display of all the options for browsing graphs in Gruff. More examples can be found on the project’s GitHub Wiki [6]. For example, one powerful option is to pick two nodes that the user thinks may be related in some way and ask Gruff to display any connections between the two nodes. Another powerful option is to select a node that is the root in a tree like structure (i.e., this doesn’t require the graph to strictly be a tree in the graph theoretic sense) and change the display to a tree layout from the selected node.

## Creating a Formal Model

# Conclusion and Recommendations

After the text edit has been completed, the paper is ready for the template. Duplicate the template file by using the Save As command, and use the naming convention prescribed by your conference for the name of your paper. In this newly created file, highlight all of the contents and import your prepared text file. You are now ready to style your paper; use the scroll down window on the left of the MS Word Formatting toolbar.

## Figures and Tables

#### Positioning Figures and Tables: Place figures and tables at the top and bottom of columns. Avoid placing them in the middle of columns. Large figures and tables may span across both columns. Figure captions should be below the figures; table heads should appear above the tables. Insert figures and tables after they are cited in the text. Use the abbreviation “Fig. 1”, even at the beginning of a sentence.

1. Table Type Styles

| Table Head | Table Column Head | | |
| --- | --- | --- | --- |
| Table column subhead | Subhead | Subhead |
| copy | More table copya |  |  |

1. Sample of a Table footnote. (*Table footnote*)
2. Example of a figure caption. (*figure caption*)

Figure Labels: Use 8 point Times New Roman for Figure labels. Use words rather than symbols or abbreviations when writing Figure axis labels to avoid confusing the reader. As an example, write the quantity “Magnetization”, or “Magnetization, M”, not just “M”. If including units in the label, present them within parentheses. Do not label axes only with units. In the example, write “Magnetization (A/m)” or “Magnetization {A[m(1)]}”, not just “A/m”. Do not label axes with a ratio of quantities and units. For example, write “Temperature (K)”, not “Temperature/K”.

# Acknowledgment

This work was conducted using the Protégé ontology editor from Stanford University. Thanks to the Protégé User Support mail list at Stanford, especially Samson . Thanks to Franz Inc. for their support with the AllegroGraph graph database. Thanks to Bob Neches for review and feedback on the project. Thanks to Robert Brulle for helping us understand the Climate Obstruction model and permission to use Figure 1 from [3].

# References

|  |  |
| --- | --- |
| [1] | J. Sterman, "Communicating climate change risks in a skeptical world," *Climatic Change,* vol. 108, no. 811, 2011. |
| [2] | S. C. Moser, "Communicating climate change: history, challenges, process and future directions," *WIREs Climate Change,* vol. 1, no. 1, pp. 31-53, 2010. |
| [3] | R. J. Brulle, J. T. Roberts, M. C. Spencer and et.al., Climate Obstruction Across Europe, R. J. Brulle, J. T. Roberts and M. C. Spencer, Eds., New York, New York: Oxford University Press, 2024. |
| [4] | A. Sheth, K. Roy and M. Gaur, "Neurosymbolic AI -- Why, What, and How," *IEEE Intelligent Systems,* 2023. |
| [5] | Franz Inc., "Neuro-Symbolic AI with AllegroGraph," Franz Inc., 2024. [Online]. Available: https://allegrograph.com/products/neuro-symbolic-ai/. [Accessed 31 July 2024]. |
| [6] | M. DeBellis, G. Gino, J. Gino and A. Balaji, "Climate Obstruction GitHub Repository," michaeldebellis.com, May 2025. [Online]. Available: https://github.com/mdebellis/Climate\_Obstruction. [Accessed 19 May 2025]. |
| [7] | Y. Gao, Y. Xiong, X. Gao, K. Jia, J. Pan, Y. Bi, Y. Dai, J. Sun, Q. Guo, M. Wang and H. Wang, "Retrieval-Augmented Generation for Large Language Models: A Survey," Corness University Preprints, 5 January 2024. [Online]. Available: https://arxiv.org/abs/2312.10997. [Accessed 19 February 2024]. |
| [8] | H. Li, Y. Su, D. Cai, Y. Wang and L. Lemao, "A Survey on Retrieval-Augmented Text Generation," 2 February 2022. [Online]. Available: https://www.semanticscholar.org/paper/A-Survey-on-Retrieval-Augmented-Text-Generation-Li-Su/e6770e3f5e74210c6863aaeed527ac4c1da419d7. [Accessed 15 April 2024]. |
| [9] | Neel Nanda et. al., "Fact Finding: Attempting to Reverse-Engineer Factual Recall on the Neuron Level," https://www.alignmentforum.org, 22 December 2023. [Online]. Available: https://www.alignmentforum.org/posts/iGuwZTHWb6DFY3sKB/fact-finding-attempting-to-reverse-engineer-factual-recall. [Accessed 26 12 2024]. |
| [10] | Z. Xu, S. Jain and Kankanhalli, "Hallucination is Inevitable: An Innate Limitation of Large Language Models. ArXiv, abs/2401.11817.," 22 January 2024. [Online]. Available: https://arxiv.org/abs/2401.11817. |
| [11] | M. DeBellis, N. Dutta, G. Gino and A. Balaji, "Integrating Ontologies and LLMs to Implement Retrieval Augmented Generation (RAG)," *Applied Ontology,* 2024. |
| [12] | C. Arellano, M. DeBellis, T. Jyothi, P. Guo, K. Kron and V. Suresh, "The DaanMatch System: Matching NGOs with with CSRs using the UN Sustainable Development Goals," in *IEEE Global Humanitarian Technology Conference (GHTC)*, Villanova University, USA, 2023. |
| [13] | A. Singhal, "Introducing the Knowledge Graph: things, not strings," Google, 16 May 2012. [Online]. Available: https://www.blog.google/products/search/introducing-knowledge-graph-things-not/. [Accessed 8 May 2023]. |
| [14] | Franz Inc., "AllegroGraph Freetext Indexing," 22 March 2023. [Online]. Available: https://franz.com/agraph/support/documentation/current/text-index.html. [Accessed 27 April 2023]. |
| [15] | D. Gulbranson, "Python Nameparser," https://github.com/, 20 September 2023. [Online]. Available: https://github.com/derek73/python-nameparser. [Accessed 19 May 2025]. |

1. Several optional matches such as for authors have been eliminated as have the prefixes in order to conserve space and because they aren’t required to understand the integration between the knowledge graph and the LLM via the Franz SPARQL “magic” property. [↑](#footnote-ref-1)