Project Report: Preventing Hallucinations in LLM QA with Knowledge Graphs

Zachary Aristei
zaristei3@gatech.edu

Rishi Banerjee
rbanerjee41@gatech.edu

Apoorva M. Krishnamurthy
apoorvamk@gatech.edu

Abstract

Large Language Models (LLMs) are a recent significant development in the field of Natural Language Processing (NLP) and have attracted considerable attention. Despite extensive research and ongoing improvements in LLMs, a notable challenge that hinders their widespread adoption is hallucination, where the model generates false or inaccurate information. In this paper we detail three different approaches that were used to reduce hallucinations in LLMs. In general in all the approaches, we only fine-tune the decoder of the language model. Before we pass in the encodings to the decoder, we mix in embeddings from the knowledge graph of entities in the query and context in various ways. Due to limited computational resources, we test our approach with the RoBERTa language model instead of an LLM and demonstrate the relative improvement in performance and show that the methods detailed can be easily extended to larger models.

1 Introduction

Hallucination is a large issue in the current generation of Large Language Models, with many active research approaches being pursued to mitigate the problem [10]. Many of these approaches focus on mitigating models themselves, without any focus on the knowledge graphs being used to train the model. The problem of hallucinations becomes doubly important when considering LLMs for professional use. The necessity of factual references is important for the usage of LLMs in the writing of documentation, and has led to issues such as in the legal profession, where a lawyer who utilized an LLM to write up a brief was censured for having nonexistent cases referenced [8]. As a result, LLMs that can utilize knowledge graphs are more trustworthy producers of documentation for any professional purpose. Knowledge graphs stand as an example of such a knowledge base, consisting of known entities that would be important to refer to and their relationships. Consequently, a question that we would like to answer is whether a graph-based model that works over such a knowledge graph can be used to check over and curate the output of an LLM so that the response of an LLM is verifiably true.

2 Related Work

The question of how to mitigate hallucinations is one that is well-tread from the literature. Past research stems from the fact of trying to find self-contradictions within the generative output of an LLM [9]. Along with this, there has also been research shown that looks to incorporate human feedback into the creation of new candidate generated output [12]. The synergies between the use of knowledge graphs and LLMs has also been explored, with both methods for the creation of knowledge graphs using LLMs and the tuning of LLM outputs with the information from knowledge graphs being proposed [11]. Other examples of knowledge graphs being used in order to improve LLMs have been shown in knowledge graph pre-trained models, which focus on making enhancements before and during training, such as optimizing the embeddings learned for each token and building separate knowledge encoders [20]. Along with this, it has also been shown that some post training methods have been used, such as generating knowledge based prompts [20]. It has also been shown in examples like KagNet that the incorporation of knowledge graphs into the inference structure of question-answering prompts could answer questions correctly with high results [6]. Knowledge graphs have also been used to modulate information retrieval for LLMs, which are tasked with question answering over the set of documents [18].

3 Data

The data that we currently intend to use for this project are going to be referencing the data that is contained within the field of general knowledge, for which we will rely on the set of relationships contained in WikiData [16]. This crowd-sourced dataset of general concepts and relationships which will comprise the knowledge graph. Of this, one of the most easily usable datasets is a scraped subset called WikiData5m [17], which contains 4.59 million entities, 20.61 million edges, and 822 types of relations. Along with this, we use a lightweight large language model in order to pass in a question with a given content and output a starting and ending position for the answer within the context. The model that we have utilized for all of our architectures has been the RoBERTa model [7], hosted on HuggingFace, which has been fine-tuned on the Stanford Question-Answering Dataset (SQuAD 2.0) [14, 13], which is based on entities contained within Wikipedia articles. Finally, in order to find a base of question-answering datasets with verified experience of hallucination, we are utilizing the HaluEval dataset [5], which provides 10,000 examples of hallucination in the question-answering task.

The architecture of a HaluEval example QA datapoint is as follows:

Question: ’Ralf Zumdick pioneered the practice of the penalty-taking-goalkeeper when he scored against a German former football goalkeeper who was in the German squad that won the 1990 what?’

Context: ’Ralf Zumdick (born 10 May 1958) is a former German football goalkeeper. Zumdick is often cited as the player who pioneered the practice of the penalty-taking-goalkeeper when he scored against Andreas Köpke in 1988.Andreas Köpke (] , born 12 March 1962 in Kiel) is a German former football goalkeeper who was in the German squad that won the 1990 FIFA World Cup, and was also part of the 1994 FIFA World Cup squad.’

Answer: ’FIFA World Cup’

Some preprocessing was necessary to extract relevant information from this data, such as the starting and ending positions of the answer that will be used in order to measure the loss and accuracy of the model outputs.

4 Methodology

We intend to build language model (LM) based model architecture, working under the assumption that methods for improving LM hallucination scales up to current LLMs. We propose to do this by embedding the graph structure of a Wikidata knowledge graph G as an input for a modified retrained language model such as BERT [1]. Using this combined representation of the knowledge graph G along with a natural language question-context-answer pair (q, c, a), the model $\theta : (G, q, c) \rightarrow a$ will be tasked with producing an output answer using the information from the graph.

Using the PyTorch Geometric implementation of the Wikidata5m dataset, we are able to extract subgraphs of the system based on preprocessing required to extract natural language entities from the questions or contexts being inputs. From this we plan to use the graph embeddings to enrich the overall encodings of the space and have the decoder model be unchanged in architecture, meaning that the calculation of loss is the same as an original RoBERTa QA model. This is calculated through the cross entropy loss of the starting and ending positions of the ground-truth answer from the decoder output.

4.1 Preprocessing

A significant challenge in the pre-processing step was to extract the entities from the natural language context and query provided that are found in the Wikidata knowledge graph. To process a given natural language string, we extract all the named entities found in the string such as names of famous personalities, countries, organisations etc as these are likely to have nodes in the Wikidata KG. Then we clean the remainder of the raw-string by removing any punctuation marks, stop-words. Finally, we tokenize the words in the string and reduce it to lower-case. To extract more entities, we form sub-strings of our processed string with contiguous sets of words starting from the longest to the smallest set of words and perform a lookup in the Wikidata KG. The idea of starting with the longest to the smallest is to get the the embeddings for whole topics, eg: "Machine Learning" instead of "Machine" and "Learning". Words found in a larger sub-string are not reused in smaller sub-strings. Finally, when doing a lookup, we use the alias to entity lookup table because a single entity in the Wikidata graph has several aliases and it is important that we not miss these entities.

4.2 Baseline Method

In order to test the effectiveness of the method, it is important to show whether the graph-enriched methods would outperform the model simply being fine-tuned on the samples with examples of hallucination. As a result, we chose to freeze all of the weights in the RoBERTa encoder, and only had the weights of the decoder evolve. The decoder, or question-answering head for this problem, was a mapping from the sentence-wise RoBERTa encodings to the estimate of the start and end positions, which are then broadcast in order to find the best start and end positions.

4.3 Hard-coded Positional and Structural Encodings

Based on the preprocessing that was performed in order to extract the entities, we were able to access the embedding of this embedding through the use of large-scale knowledge graph embedding schemes such as SimplE [4]. The question then becomes how to incorporate these embedding into the model. In order to utilize the embeddings of the entities to enrich the overall model. The set of tokens that correspond to the set of embeddings thus needed to be found, and was found during the preprocessing step. Thus, at the RoBERTa encoding positions, the corresponding graph encoding was concatenated at the following positions. For all positions for which there was not a mapped entity, a zero vector of the same size was concatenated. In order to return these encodings to the same size for an unchanged question-answering head, a fully-connected layer was added in order to map the concatenated encodings to the size of the original RoBERTa encodings.

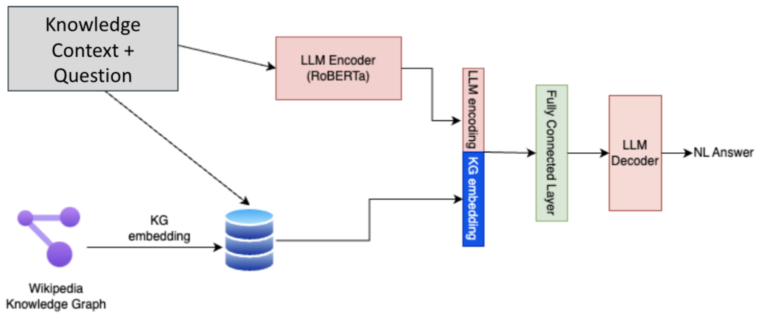

Figure 1: Baseline Architecture

4.4 Subgraph GNN Method

This method was used in order to build curated representations from the graph using a graph neural network trained on the HaluEval data. We also wanted for the model to show more adaptively learn the relations between the information encoded in the graph and the positions of the questions and the context.

As a result, the construction of the model first needed to retrieve a subgraph of the overall knowledge graph in order to derive a general embedding for the overall features and structure of the graph. In this graph, we were able use the previous preprocessing in order to select the overall entities that were involved in the question and context. Next, a k-hop subgraph was formed between the nodes. These were passed in along with the question and context. The GNN was then outputting an embedding for the subgraph.

The architecture of the GNN model is shown in 2. We utilized an Attention-Based layer to encode relationships of nodes over the entire graph [15], which encodes the edge attributes as well. Next, we used mean pooling which results in one global node embedding and lastly a linear readout to control the final size of the graph embedding.

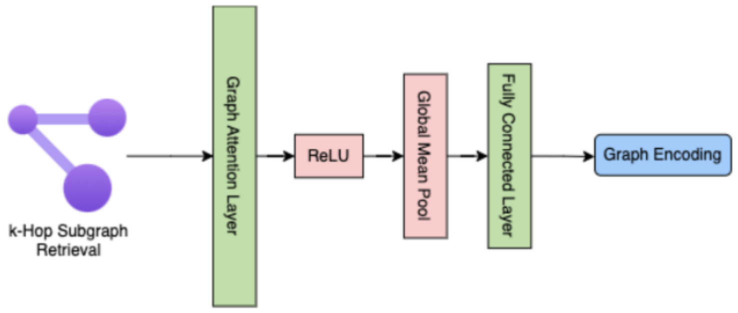

Figure 2: GNN Architecture

In order to enrich the position-wise encodings with these graph embeddings, we first broadcast this embedding over all positions and then applied a fully connected layer in order to build a representation that can enrich the position-wise encodings.

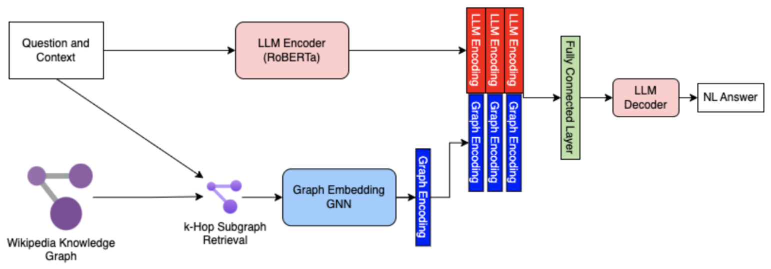

Figure 3: Architecture for Subgraph Enrichment

4.5 Nearest-Neighbors Based Methods

In this method, we decided to build upon the hard-coded positional and structural embeddings method. While, the QA task is extractive in nature, we believe that augmenting the encoder outputs with the KG embeddings of other closely related entities will nudge the LLMs answer in the right direction and prevent hallucination. This method is similar to the previous method, where we extract the relevant sub-graph but instead of using the k-hop neighbourhood, we find a set of knowledge graph embeddings that have the minimum cosine distance to the entity embedding at hand.

This is driven by the intuition that the Knowledge graph embeddings implicitly capture the graph structure and hence nodes that are close in the graph have similar embeddings. Another strength of this approach, is that with the increasing efficiency of vector databases, it easier to lookup the similar KG embeddings as opposed to performing a graph traversal to find the nearest neighbours when answering queries in real-time.

To speed up our training process, we create sets of entities found in the question and context along with another set consisting of their nearby entities. We do this by building a vector database of the SimplE KG embeddings that we have and performing a lookup to get the nearest 30 embeddings for each of the entities in our question. We use the the VDBLite python library built on a FAISS[2] core which is optimised for dense vector lookup and clustering.

The model used to incorporate these embeddings is similar to our initial model. We simply average all the nearest entities' embeddings and use a fully connected layer to mix this in after we have obtained the enriched encodings same as what we obtained in the first method. We reason that this model should perform at least as well as our first model as the final fully connected layer can assign 0 weights to the averaged neighbouring entities to obtain a equivalent model.

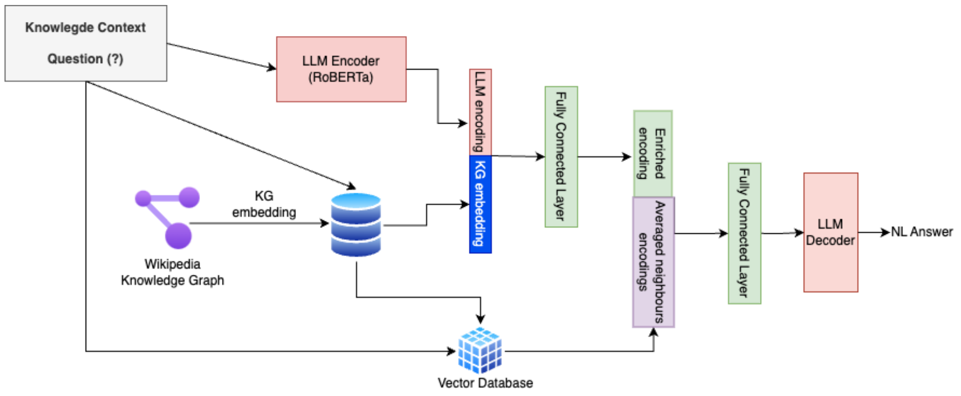

Figure 4: Nearest Neighbors-based Architecture

4.6 Evaluation Metrics

The evaluation metrics used are tied into the returned estimate of the start and end positions of the model. This model utilizes the set of words that correspond to the answer to create two metrics; exact match, which finds the proportion of the examples that correctly map to the correct set and the F1-score, which finds the precision of the overlapping sets of words between the ground truth and the prediction. These metrics are classic methods used to benchmark the accuracy of models in question-answering tasks [14].

5 Experiments

For each of the methods described above, we utilized HuggingFace transformers' `trainer` module [19] for 20 full epochs of the dataset with 80% training data. For each of these methodologies, we compute the exact-match accuracy and f1 scores when compared to baseline RoBERTA, as well as evaluate the valuation loss. The results for each model can be seen below:

Method	Evaluation Loss	Exact Match Accuracy	F1-Score
Baseline RoBERTa	1.012	.6355	.7292
Hard-coded Encodings	1.041	.6245	.7204
Subgraph GNN	1.059	.6105	.7103
Nearest Neighbors	.9957	.7050	.7954

Table 1: Evaluation losses and other metrics after 20 epochs

From our analysis, we notice that the model that performs the best is our nearest neighbors classifier, which aggregates the vectors together to form a representation for the whole knowledge graph which was a surprising finding for us. The fact that the other methods did not deviate significantly from the baseline RoBERTa values in comparison indicates that decoupling the ordering of the graph embeddings from the orderings in the prompts is critical in order to efficiently utilize the knowledge graph embeddings after many training epochs. We notice that our knowledge graph-based models

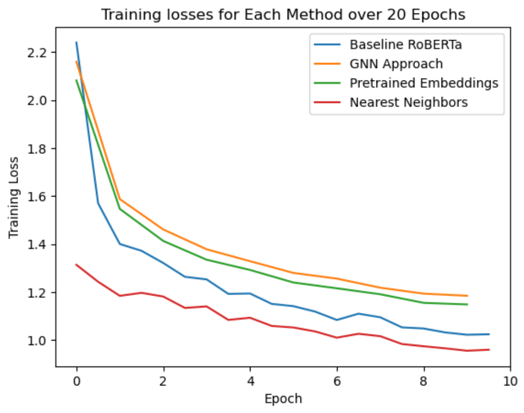

Figure 5: Training Losses over 10 Epochs

are able to out-perform the baseline model by choosing smaller subsets on examples where baseline RoBERTa is more likely to over answer, similar to hallucination, like in the following example:

Question: 'Who magazine is based in Milan, Italy Donna Moderna or Parents?'

Context: 'Parents, published by Meredith Corporation, is an American mass circulation monthly magazine that features scientific information on child development geared to help parents in raising their children.Donna Moderna (meaning "Modern Woman" in English) is an Italian language weekly women's and fashion magazine based in Milan, Italy.'

Answer: 'donna moderna'

Nearest Neighbors Model Prediction: 'donna moderna'

RoBERTa Prediction: 'donna moderna (meaning "modern woman" in english) is an italian language weekly women's and fashion'

6 Conclusion

What we see above is that the inclusion of knowledge graph embedding info in the context of language models does significantly boost performance in terms of question answering models. But since each of these methods for incorporating the knowledge graph occurs before the decoder portion of the model's architecture, these methods are entirely compatible with different decoder architectures and training loss parameters. For instance, this method can easily be applied to LLM architectures that utilize a Reinforcement Learning with Human Feedback (RLHF) in their training loss.

Incorporating these methods into an RLHF system or open-ended LLM architecture provides a promising next step for where to take investigation of these methods, and would've been our next step should we have had the time and compute to do so. In addition, there is active research in incorporating knowledge graph embeddings more robustly into the encoder architecture. For instance, Kang et. al propose an embedding scheme for graph relations that preserves both permutation and relation inversion invariance for the graph nodes [3]. This method provides a logical next step for enhancing our encoding methodology while preserving the unique encodings for each graph vertex.

One existing drawback with our approaches is the amount of preprocessing compute that is required to construct a knowledge graph, or extract a relevent subgraph based on the natural language context. This may make the methods infeasible for use within the context of a edge inference use case. However, for LLM foundation models, it is assumed that the computational cost for maintaining and querying a large knowledge graph is affordable relative to the already expensive computational cost of maintaining a large foundation model with millions of weights. Thus, we are optimistic that methods similar to the ones we explored will be helpful in future research for reducing inaccuracy and hallication in future LLM applications.

7 Contributions

- Literature Survey - Rishi, Apoorva, Zachary
- Dataset Collection and Pre-processing - Apoorva
- Model design - Rishi, Apoorva, Zachary
- Baseline method implementation - Rishi
- Hard Coded Positional and Structural Encoding Implementation - Apoorva
- Subgraph GNN Model Implementation - Rishi
- Nearest-Neighbors Based Method - Apoorva
- Model training and finetuning - Zachary
- Ablation Studies and Experiment Data Collection - Zachary
- Results and Analysis - Zachary
- Report and presentation - Zachary, Apoorva, Rishi

8 Link to Code

Our codebase can be found at https://github.com/apoorva-mk/KG-LLM-Hallucination

References

[1] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[2] Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7(3):535–547, 2019.

[3] Minki Kang, Jin Myung Kwak, Jinheon Baek, and Sung Ju Hwang. Knowledge graph-augmented language models for knowledge-grounded dialogue generation, 2023.

[4] Seyed Mehran Kazemi and David Poole. Simple embedding for link prediction in knowledge graphs. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 4289–4300, Red Hook, NY, USA, 2018. Curran Associates Inc.

[5] Junyi Li, Xiaoxue Cheng, Wayne Xin Zhao, Jian-Yun Nie, and Ji-Rong Wen. Halueval: A large-scale hallucination evaluation benchmark for large language models, 2023.

[6] Bill Yuchen Lin, Xinyue Chen, Jamin Chen, and Xiang Ren. Kagnet: Knowledge-aware graph networks for commonsense reasoning, 2019.

[7] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach, 2019.

[8] Sara Merkin. New york lawyers sanctioned for using fake chatgpt cases in legal brief. 2023.

[9] Niels Mündler, Jingxuan He, Slobodan Jenko, and Martin Vechev. Self-contradictory hallucinations of large language models: Evaluation, detection and mitigation, 2023.

[10] OpenAI. Gpt-4 technical report, 2023.

[11] Shirui Pan, Linhao Luo, Yufei Wang, Chen Chen, Jiapu Wang, and Xindong Wu. Unifying large language models and knowledge graphs: A roadmap, 2023.

[12] Baolin Peng, Michel Galley, Pengcheng He, Hao Cheng, Yujia Xie, Yu Hu, Qiuyuan Huang, Lars Liden, Zhou Yu, Weizhu Chen, and Jianfeng Gao. Check your facts and try again: Improving large language models with external knowledge and automated feedback, 2023.

[13] Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for squad, 2018.

[14] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text, 2016.

[15] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2018.

[16] Denny Vrandeić and Markus Krötzsch. Wikidata: a free collaborative knowledgebase. *Commun. ACM*, 57:78–85, 2014.

[17] Xiaozhi Wang, Tianyu Gao, Zhaocheng Zhu, Zhengyan Zhang, Zhiyuan Liu, Juanzi Li, and Jian Tang. Kepler: A unified model for knowledge embedding and pre-trained language representation, 2020.

[18] Yu Wang, Nedim Lipka, Ryan A. Rossi, Alexa Siu, Ruiyi Zhang, and Tyler Derr. Knowledge graph prompting for multi-document question answering, 2023.

[19] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, October 2020. Association for Computational Linguistics.

[20] Linyao Yang, Hongyang Chen, Zhao Li, Xiao Ding, and Xindong Wu. Chatgpt is not enough: Enhancing large language models with knowledge graphs for fact-aware language modeling, 2023.