Enhancing Explanations and Contextual Reasoning of GPT Models Using Knowledge Graphs

Jacob Yoder

November 21, 2024

Abstract

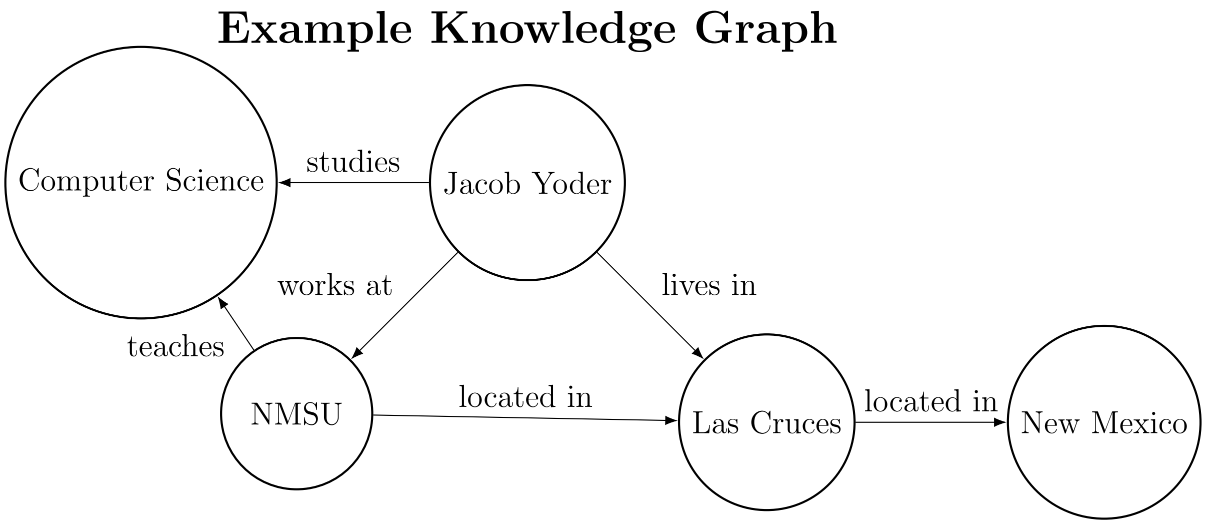
Generative Pre-trained Transformer (GPT) models have revolutionized natural language processing by excelling in memory-based and generative tasks. However, their limitations in reasoning and providing interpretable explanations hinder their reliability in critical applications and domain specific topics. This paper offers the use of Knowledge Graphs as a way to address these issues, offering two distinct improvements: improved contextual awareness through Knowledge Graph retrieval, and the ability to trace reasoning paths using the graph's structure as a form of explainability. Results demonstrate that leveraging Knowledge Graphs allows for context bolstering improving on traditional Retrieval Augmented Generation, as well as a way for information generated by GPTs to be verified. The study concludes with a discussion on trade-offs and potential future directions, including the use of weighted graph connections to further refine efficiency and relevance.

1 Introduction

1.1 Knowledge Graphs

Knowledge Graphs (KGs) are structured representations of knowledge that capture entities (e.g., people, places, concepts) as nodes and their interrelations (e.g., part_of, works_at, lives_near) as edges. Unlike traditional data structures, such as relational databases or hierarchical trees, KGs provide a flexible and intuitive way to model complex, interconnected data.

Each entity in a KG is often enriched with attributes or metadata, such as names, dates, or descriptions. Relationships between entities carry semantic meaning that encodes how entities relate to one another. For example, in a KG for scientific research, a node representing a "researcher" might have edges to nodes representing their "affiliations," "published papers," and "collaborators."



KGs are widely utilized in domains such as:

- **Natural Language Processing (NLP)**: Enabling semantic search, entity disambiguation, and contextual understanding.
- **Recommender Systems**: Enhancing recommendations by identifying relationships between users, items, and preferences.
- **Medical AI (MedAI)**: Supporting clinical decision-making by linking diseases, symptoms, treatments, and outcomes into a coherent graph of medical knowledge.
- **Social Media**: Finding groups with shared interests, hobbies, and beliefs is a common marketing goal within social graphs.

Their ability to organize and query vast amounts of interconnected data makes KGs a powerful tool for improving reasoning, context, and knowledge retrieval in AI systems.

1.2 Large Language Models

Large Language Models (LLMs) are advanced AI systems trained on massive amounts of text data to perform a variety of language tasks, such as text generation, question answering, summarization, and more. These models, typically based on transformer architectures, leverage billions of parameters to capture patterns, relationships, and context from text data.

Despite their versatility, LLMs face significant challenges:

- **Reliability**: LLMs may generate plausible-sounding but incorrect or misleading information, often called "hallucinations."
- **Domain Specific Understanding**: Pre-trained LLMs, like a GPT, may lack the specialized knowledge required for niche or highly technical domains without costly and resource-intensive retraining.

1.3 Generative Pre-trained Transformers (GPTs)

Generative Pre-trained Transformers are a family of LLMs based on the transformer architecture. While the terms GPT and LLM are used somewhat interchangeably in this paper, this research specifically focuses on GPT models. They operate in two main phases:

- **Pre-training**: The model learns to predict the next word in a sentence using vast amounts of training text data. This phase allows GPT to capture general language patterns, semantics, and structure.

- **Fine-tuning (optional)**: For more specific tasks or domains, GPT can be fine-tuned on smaller, specialized datasets, aligning the model to task-specific objectives (e.g., summarization or code generation).

GPT models are widely recognized for their generative capabilities, producing coherent and contextually relevant text. However, they rely heavily on the statistical patterns of their training data and lack inherent reasoning or verification mechanisms. They also cannot learn new information without re-training, which is an expensive and time consuming process. This research seeks to address some of these limitations by integrating Knowledge Graphs with GPT models, proposing a framework to enhance their contextual reasoning and explainability.

1.4 Research Objectives

This research addresses the following key questions:

- How can we improve the reliability and trustworthiness of LLMs like GPT?

- How can we solve the lack of domain-specific knowledge without retraining a model?

By leveraging the structured and interconnected nature of Knowledge Graphs, this work proposes a framework to enhance the contextual reasoning and explanations provided by GPT models, enabling them to generate more accurate, verifiable, and transparent responses.

2 Background

The integration of knowledge retrieval with Large Language Models (LLMs) has been the focus of several notable studies, each addressing specific challenges and limitations:

- **Retrieval-Augmented Generation (RAG)**: RAG combines LLMs with external document retrieval systems, allowing the model to ground its responses in relevant documents [5]. While effective for many knowledge-intensive tasks, traditional RAG systems inherently lack the ability to model complex relationships between retrieved documents or entities. This limitation restricts their capability to provide deeper reasoning or multi-step connections.

- **Harvard MedAI**: This system leverages domain-specific Knowledge Graphs (KGs), specifically in the field of medicine, to enhance clinical decision-making and diagnosis. By structuring information about diseases, symptoms, and treatments, MedAI allows graph traversal to retrieve highly relevant and interconnected knowledge [11]. However, its domain specificity limits its applicability to general settings. While it demonstrates significant improvements for KG-based models, it fails to provide reasoning and explainability.

- **LightGraph**: LightGraph introduces a lightweight approach to combining Knowledge Graphs with RAG systems, prioritizing efficiency and scalability [12]. While it achieves faster retrieval times, this comes at the cost of reduced detail and relational complexity, making it less suited for reasoning-intensive tasks.

Efforts to enhance explanation generation for LLMs have also gained traction:

- **XplainLLM**: XplainLLM primarily introduces a dataset designed to enhance LLM explainability by attaching reasoning to model outputs. While the paper provides a methodology for generating explanations, its focus lies more on using this dataset of explanations to find a best fitting explanation then alter it to fit the specific question. Although this technique is effective for generating explanations, it overlooks the potential for constructing logical reasoning paths, instead relying on pre-existing examples without explicitly tracing the reasoning for a specific question/answer pair.

This paper builds on concepts from modern RAG systems by integrating Knowledge Graph-based search strategies for contextual enhancement. This paper introduces a new method for explanation generation that leverages the same Knowledge Graph used for contextual bolstering. By constructing coherent reasoning paths, the proposed framework not only retrieves relevant information but also explains how answers are derived, integrating two emerging approaches for enhancing LLM responses.

3 Methods

3.1 Overview of Framework

The proposed framework in this paper includes both the KG-RAG system to bolster the context of the GPT, as well as using the same Knowledge Graph for a reasoning generation. The visual representation of the complete flow can seen in Figure 1, and an output example can be seen in Listing 1 in the figures section of the paper.

3.2 Knowledge Graph Construction

Two approaches were considered:

- Using pre-existing KGs tailored to specific domains (e.g., medical KGs).

- Creating KGs from raw, unstructured data using NLP pipelines.

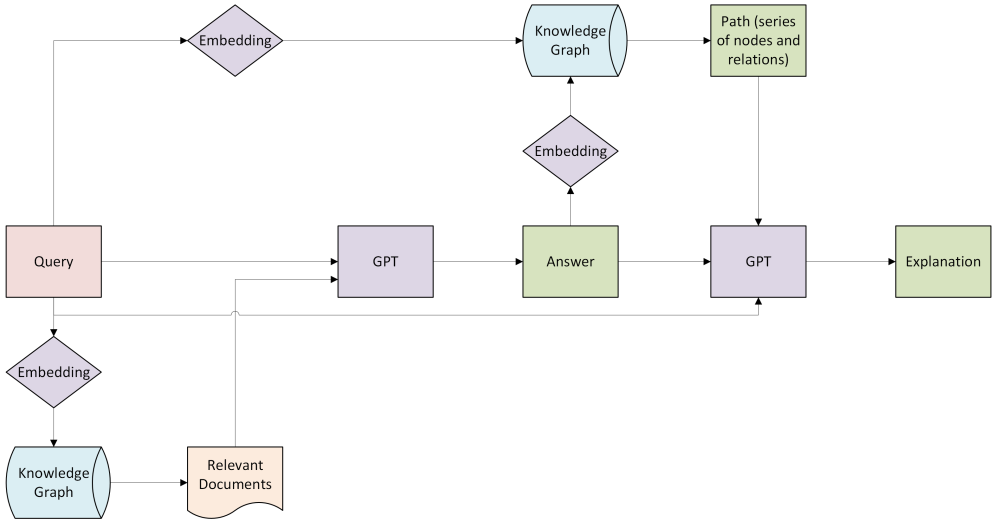


Figure 1: The framework starts with a query, using this to find relevant information from the KG. An answer is then generated by the GPT model, and a reasoning path is constructed within the KG to semantically link the query to the answer. Then the path, relevant documents, query, and answer can be used for the explanation generation.

The simplest choice is clearly to use a pre-existing KG for your specific topic and alter it to work with your algorithms. However, this is not an option in many cases because finding a pre-made KG with all the needed information is nearly impossible, however it was a great place to start and learn what is useful to a graph.

Thus, when tasked with creating a large, domain-specific KG, the only practical solution is to generate it programmatically. There are a number of ways to take a piece of text and create a KG from it [2][3][6], and is quite a well studied problem. The two options that I considered were splitting a text into sentences and using a pattern based approach to split the sentence (i.e. subject, verb, object). This approach however faces a number of issues, mainly the lack of flexibility in setting these rules and semantic and syntactic ambiguity. So, I decided on using an LLM to convert paragraphs at a time into a KG. This option allows for more flexibility, speed from doing large chunks at a time, and can be fine-tuned for better accuracy. Pseudo-code for this method of KG construction can be seen in Algorithm 1.

Some challenges encountered with this method include the issue of the LLM creating multiple nodes for the same topic. When something has multiple names or appears in different contexts within the text, the LLM occasionally produces distinct nodes with differing labels for the same concept. For example, in this paper, “Knowledge Graph” is introduced early, but later, the term “KG” is used. The LLM may not recognize these as identical due to their differing labels. However, the algorithms used in this paper rely on the relevance of the text associated with each node rather than the labels exclusively. As a result, when searching for relevant nodes, either multiple nodes for the same topic may appear, or some may be excluded. Regardless, the algorithm prioritizes the most relevant nodes, ensuring the overall outcome is unaffected by the duplicate nodes in the graph.

Another potential issue to consider is the risk of introducing malicious or incorrect documents into the system that generates the KG. Since the KG is treated as a ground truth

Algorithm 1 Knowledge Graph Generation

```
Split text document into chunks
Create graph object: G
for all chunk in chunks do
    Convert chunk to small KG: k
    Add k to G
end for
for all node in G do
    Combined document list Docs
    for all doc_reference in node do
        add doc_reference to docs
        Create embedding of docs: embedding
        Add embedding to node
    end for
end for
Save the graph G to .json
Return complete graph G
```

during searches, the presence of "bad" data can lead to flawed or misleading answers. This problem underscores a fundamental challenge in the design of LLMs and AI systems: they are only as reliable as the data they are built upon. Without mechanisms to verify the accuracy of the input data, the system remains vulnerable to such issues. Unfortunately, this problem is a long-standing challenge in AI and still awaits a Nobel-winning solution. For the purposes of this work, it will be assumed that the data used to generate the KG has been thoroughly vetted and is free from errors.

3.3 Knowledge Graph Search

Three search methods were explored:

1. **Topic Search**: Extract topics from queries and answers, matching them to nodes in the KG.
2. **Keyword Search**: Match topics and referenced text; slower but higher precision.
3. **Embedding Search**: Use average embeddings of references and nodes to enable semantic matching.

3.3.1 Topic Search

The idea behind topic search was to identify the main idea, or topic, of a query, and then locate the relevant documents by finding the associated node in the KG. The documents used to create that node would then serve as the context for the query.

This simple approach allows for efficient lookup of the KG by checking if a node with the same name as the topic exists, which operates with a complexity of O(1). However, this method often misses topics or fails to find a node when the topic either does not exist in the KG or is represented by a different name.

To address this issue, a Word2Vec database was integrated to allow searching the KG using a topic's vector representation and finding the most closely related topic based on semantic similarity. While effective, this method faces a similar challenge: if the topic or node name is missing from the Word2Vec database, it results in failure to locate relevant nodes.

Pros: Efficient lookup, simple implementation, and scalable.

Cons: Over-simplistic, brittle to variations, inflexible, and missing contexts.

3.3.2 Keyword Search

Improving on the simplistic approach of topic searching, I implemented a traditional RAG method of keyword searching. This method extracts the most important topic or keyword from the query using topic extraction models or an OpenAI API call. The extracted keyword is then used to search all documents associated with each node in the KG, evaluating their relevance based on the frequency of the keyword.

Pros: Improved document matching, more flexibility, and simple.

Cons: Computationally expensive, keyword dependency, and lack of semantic understanding.

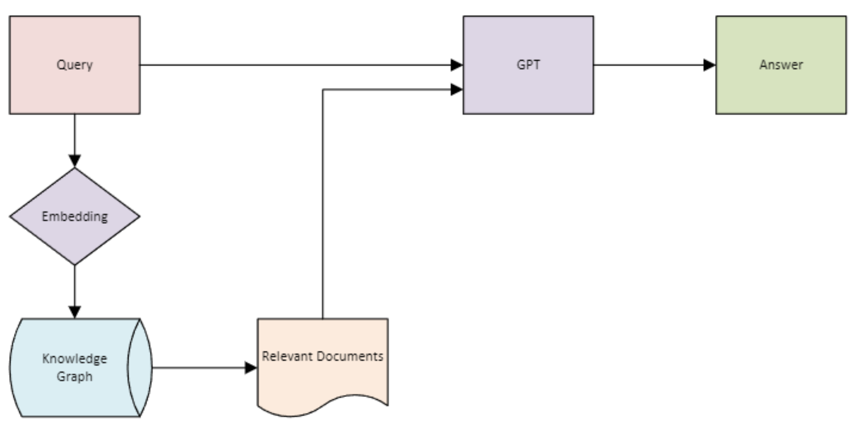


Figure 2: A query is first converted into a vector representing its semantic content. The KG is then searched for nodes with the most similar semantic meaning based on this vector. Once identified, relevant documents associated with these nodes are extracted and used as context for the query, enhancing the GPT's response generation.

3.3.3 Embedding Search

This final method of searching has proven to be the most reliable, leveraging functions from GPT itself—embeddings. When building the KG, the documents referenced by each node are combined and converted into a vector representation using the OpenAI embedding API. This vector encapsulates the semantic meaning of the text. As shown in Figure 2, a query can then be converted into a vector using the same embedding system, and the query vector is compared to the node vectors to identify the most semantically related nodes and their associated documents.

Pros: Improved accuracy, semantic understanding, and robust.

Cons: Initial computational cost, dependency on pre-trained models, and scaling difficulties (discussed more in future work).

3.4 Reasoning Path Generation

Finding the reasoning for a GPT is, in actuality, a nearly impossible task. GPT models are trained on large datasets, and their outputs are generated based on the patterns and relationships learned during this training process. Or in other words, a GPT simply responds with the most likely next words based on it's training data. This means that there is no logic, thought, or reasoning used by a GPT when producing an answer. However, we can estimate the reasoning that a human may take to get from a question to an answer using KGs.

This is done by taking a query and a corresponding answer from a GPT, and finding the most similar nodes in the KG, then finding the most logical path between them. This can be done by finding the least amount of jumps or using a function to approximate the "cost" of moving from one node to another. In this paper, I utilized the A* search algorithm and developed a custom heuristic function to estimate the "cost" of selecting one node over another in the reasoning path. This heuristic is based on cosine similarity, measuring the semantic difference between node embeddings.

3.4.1 A* Algorithm

The A* algorithm is a well know graph traversing algorithm that uses a heuristic to determine which nodes should be searched next for finding a near optimal path. In this case, I used cosine similarity between the two nodes embedding to evaluate the cost of making the jump from node 1 to node 2.

An A* search using a similar heuristic mimics human cognitive processes by connecting unknown or new ideas to familiar ones through small, incremental steps. The heuristic prioritizes topics or nodes that are closely related to the starting point, assigning them a lower cost. At the same time, it evaluates whether the path is moving in the general direction of the goal by calculating the estimated cost from each potential next node to the target node. This combination of local relevance and global direction allows the search to find a path that is both logical (i.e., small connected steps) and near optimal.

3.5 Human Readable Reasoning

We now have a path of nodes and relationships between these nodes to represent a path of reasoning, however this can be difficult to read and understand. An example given the query "Why does abstract art evoke different emotional responses in viewers?" could look like this:

Algorithm 2 Cosine Similarity for Node Embeddings

Require: Two node embeddings $\mathbf{e_1}$ and $\mathbf{e_2}$, where each embedding is a vector.

Ensure: A cost estimation $S \in [0, 2]$ that evaluates how similar the nodes are.

1: **Step 1:** Compute the dot product of $\mathbf{e_1}$ and $\mathbf{e_2}$:

$$\text{dot\_product} \leftarrow \sum_{i=1}^{n} \mathbf{e_1}[i] \cdot \mathbf{e_2}[i]$$

2: **Step 2:** Compute the magnitude (norm) of each embedding:

$$\text{norm1} \leftarrow \sqrt{\sum_{i=1}^{n} (\mathbf{e_1}[i])^2}, \quad \text{norm2} \leftarrow \sqrt{\sum_{i=1}^{n} (\mathbf{e_2}[i])^2}$$

3: **Step 3:** Calculate the cosine similarity:

$$S \leftarrow \frac{\text{dot\_product}}{\text{norm1} \cdot \text{norm2}}$$

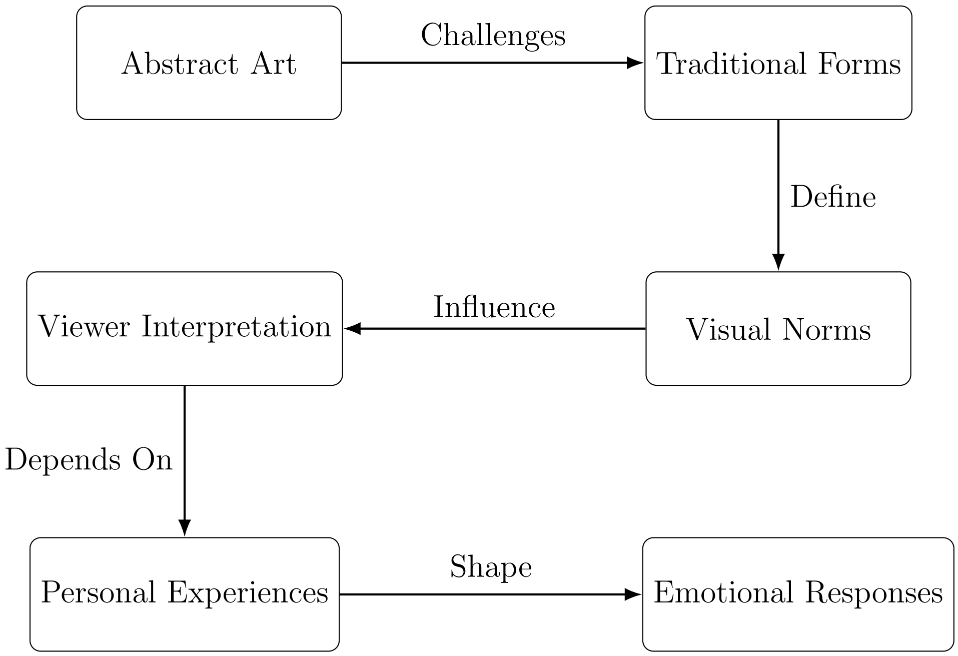
4: **Step 4:** Convert S to a "cost" evaluation:

$$S \leftarrow (1 - S)$$

5: **Step 5:** Interpret S:

$$S \in [0, 2], \quad \text{where:}$$

- $S = 0$: The vectors $\mathbf{e_1}$ and $\mathbf{e_2}$ are pointing in the same direction.
- $S = 1$: The vectors $\mathbf{e_1}$ and $\mathbf{e_2}$ are orthogonal, indicating no similarity.
- $S = 2$: The vectors $\mathbf{e_1}$ and $\mathbf{e_2}$ are pointing in opposite directions.
- $S \in (0, 2)$: The similarity score reflects the degree of alignment between the vectors, with lower values indicating greater similarity.



To address this issue, I implemented a method where the path, along with relevant documents from each node and the query/answer pair, is fed into a GPT. The GPT is then prompted to generate an explanation for the answer based on the provided context. Specifically, the model is instructed to follow the given path and use the associated information to explain how the path would serve as a reasoning process.

I acknowledge that there is an inherent irony in relying on an GPT—which itself produces responses that we cannot entirely trust or accept as factual—to explain the reasoning of another GPT. However, the purpose of this explanation generation is not to guarantee the correctness of the answer but to provide a tool for verification. Evaluating any answer from a GPT requires comprehensive understanding of the topic and its related concepts. Yet, possessing such complete understanding would negate the need for the GPT in the first place.

This solution bridges the gap by offering a way for someone without complete expertise to trace the logical path used by the LLM to arrive at the answer. If the reasoning path deviates, contains inconsistencies, or makes an unsupported leap, these flaws can be identified and the answer discarded. This process provides a means for users to assess the reasoning behind a GPT's response without needing to fully comprehend the entire domain themselves.

4 Results

4.1 Search Performance

- **Topic Search**: Fast but limited in capturing nuanced relationships.
- **Keyword Search**: High precision but computationally expensive.
- **Embedding Search**: Balanced precision and efficiency, as well as offered semantic comprehension.

Ultimately, embedding search proved to be the only viable option. Its ability to semantically comprehend queries and locate the most relevant documents, regardless of variations in phrasing, made it far superior to the other methods. Unlike keyword search, which became extraordinarily slow with larger graphs, embedding search remained flexible and efficient, ensuring scalability and reliability across diverse queries. A comparison between an unaltered GPT, RAG GPT, and KG-RAG can be seen in Table 1.

4.2 KG-RAG Integration

- **Pre-embedding KG-RAG**: This approach struggled with search efficacy as it relied on graph traversal without semantic understanding. It often prioritized the shortest path over the most logical or contextually relevant one, leading to suboptimal document retrieval.

- **Post-embedding KG-RAG**: By integrating embeddings into the Knowledge Graph, this method significantly improved search quality. It enabled the identification of reasoning paths and documents that aligned more closely with the semantic intent of the query.

When compared to a traditional RAG system, the KG-RAG produced nearly identical sets of documents. This outcome is expected, as the core mechanism driving the KG-RAG is embedding-based search, which aligns closely with the methodology used in modern RAG systems. However, the KG-RAG provides additional advantages by leveraging the structured relationships within the Knowledge Graph, enabling not only document retrieval but also relational information that traditional RAG systems lack.

4.3 Reasoning Path Analysis

Embedding-based pathfinding significantly enhanced the representativeness of reasoning paths by leveraging semantic understanding. The use of embeddings enabled cosine similarity to approximate the semantic differences between nodes, allowing the pathfinding algorithm to prioritize topic similarity rather than merely minimizing the number of jumps. This approach ensured that the paths generated were more contextually relevant, offering a more reasonable explanation of logical steps taken.

5 Discussion

The integration of Knowledge Graphs into GPT workflows demonstrated:

- Enhanced Reasoning and Explanation Quality: By leveraging the structured relationships within Knowledge Graphs, GPT workflows were able to generate explanations that were more coherent and logically grounded. This was particularly useful in tasks requiring step-by-step reasoning, such as answering complex questions, generating multi-hop explanations, or solving problems in technical domains.

- Trade-offs Between Precision and Computational Cost: The choice of search method (e.g., topic-based, keyword-based, or embedding-based) influenced both the precision of retrieved documents and the system's computational efficiency. Embedding-based search provided the best balance, offering high semantic understanding at a reasonable computational cost, making it suitable for dynamic and context-heavy applications.

The implementation of Knowledge Graphs in GPT workflows offers numerous benefits including: enhancing reasoning, context, and explanation generation. This approach has significant potential in fields such as education, customer support, healthcare, and legal analysis, where complex and interconnected information needs to be processed and explained effectively. As demonstrated in previous works, Knowledge Graphs effectively represent complex information through connections and relationships. By leveraging this data representation, GPTs can be altered with domain-specific knowledge without requiring expensive retraining or fine-tuning, making them more adaptable to specialized tasks.

5.1 Future Work

There are a number of places for improvement within the complete flow:

1. A potential direction for future work could explore directed search rather than exhaustive graph searching. One advantage of Knowledge Graphs is that similar topics are located close to one another, meaning if we can find a similar topic when searching, there is no real need to search the whole graph, we can just search locally and with high likelihood find the most relevant nodes without a complete search. Significantly improving search speed in large KGs with minimal, if any, compromise on efficacy.

2. Building on the previous idea, implementing a weighted graph could improve search speed by assigning a strength to every relationship and filtering out weaker relationships. This strength could be determined by the model's confidence in that connection being valid or by the frequency a connection has been made between those nodes across all documents. This would work similarly to embeddings, by leading the search algorithm to related nodes, but could offer greater searching speeds and reliability.

3. Knowledge Graph creation could also be improved by consolidating nodes with similar topics but differing labels. For example "KG" and "Knowledge Graphs" represent the same idea, but since they have different labels, they would often be treated as unique nodes in the KG. While this shouldn't have any real effect on either of the two proposed LLM improvements, it may be a better way to consolidate information and allow for better visualization of the graph as a whole. By utilizing a semantic similarity comparison (like cosine similarity), nodes could be programmatically evaluated for similarities. Nodes exceeding some threshold could then be merged into a single node with multiple aliases.

4. A significant improvement to the KG would involve integrating the document chunks directly into the graph. Currently, the document chunks are stored separately and accessed through references in each node. Simplifying this process by combining all

information within the graph would reduce complexity. Additionally, this approach enables the pre-computation of embeddings for each document chunk. Node embeddings could then be calculated as an aggregation of the embeddings of their connected documents. This strategy would significantly reduce the computational cost of scaling the KG, as each document chunk would only require embedding computation once, regardless of how many nodes reference it.

6 Code

The code for this work is available on GitHub at `https://github.com/Novaii-Yoder/KnowledgeGraphResearch`. The repository includes a couple Python library files that implement the methods described in this paper. Additionally, it contains an example Jupyter Notebook demonstrating how to use the libraries and visualize your own results.

7 Figures

Feature/Aspect	**Normal GPT Call**	**RAG Call**	**KG-RAG Call**
What is NAMAC?	NAMAC stands for the National Alliance for Media Arts and Culture. It is an organization that supports and advocates for media artists and organizations.	NAMAC stands for Nearly Autonomous Management and Control system. It is a comprehensive control system developed for advanced reactors that assists in plant operations by providing recommendations to operators. The system is designed with a three-layer architecture that includes a knowledge base, a Digital Twin developmental layer, and a NAMAC operational layer. NAMAC combines different functions and techniques to create a structured workflow for promoting safety and efficiency in reactor operations.	NAMAC stands for Nearly Autonomous Management and Control system. It is a system developed for advanced reactors that aims to provide recommendations to operators for maintaining the safety and performance of the reactor. NAMAC consists of three layers: the knowledge base, the Digital Twin developmental layer, and the NAMAC operational layer. The Digital Twins (DTs) in the system acquire knowledge from the knowledge base to support different functions such as diagnosis, strategy planning, prognosis, and strategy assessment. The system operates based on real-time measurements and data learning from the DTs.
Who developed EMDAP?	I'm not sure, would you like me to look it up for you?	EMDAP was developed by the U.S. Nuclear Regulatory Commission (NRC).	EMDAP was developed by the U.S. Nuclear Regulatory Commission (NRC).

Table 1: Comparison of GPT 3.5 turbo, GPT 3.5 turbo with RAG, and with KG-RAG. The RAG systems used the same 5 research papers as input documents from the NAMAC project from North Carolina State University.

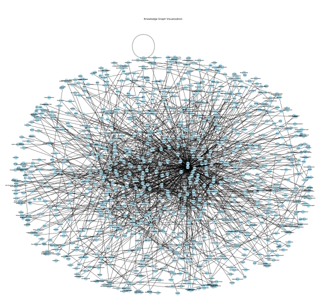


Figure 3: A KG generated from the full text of 1984 by George Orwell.

Listing 1: Example Output

```
Query:
   How does the Party influence societal control and enforce its ideology?

GPT Answer:
   The Party in the described society influences societal control and enforces its ideology
       through various means. These include constant surveillance of citizens, manipulation
       of public opinion through the media like print, film, radio, and television, ensuring
        complete obedience to the State, enforcing uniformity of opinion on all subjects,
       maintaining a hierarchical society where the ruling group perpetuates itself through
       a specific world-view and way of life, and eliminating any dissent or deviation of
       opinion, even on minor issues, among its members. Additionally, the Party
       concentrates property in the hands of a select group, controls all aspects of society
       , and maintains a collectivist structure to solidify its power and ensure economic
       inequality remains permanent. The Party's control is also reinforced by the use of
       the Thought Police to eliminate potential sources of discontent or opposition.

Path:
   [('Catholic Church', 'compares_with', 'Party'), ('Party', 'utilizes', 'Ministry of Truth
       '), ('Ministry of Truth', 'related_to', 'Ministry of Peace'), ('Ministry of Peace', '
       work_for', 'Scientists'), ('Scientists', 'includes', 'New Aristocracy'), ('New
       Aristocracy', 'includes', 'Journalists')]

Explanation:
   The path of relationships in the context leads us to understand how the Party influences
       societal control and enforces its ideology. The connection between the Catholic
       Church and the Party highlights a comparison in terms of influence and power. This
       comparison suggests that the Party wields a significant amount of control similar to
       that of a powerful institution like the Catholic Church.

Further down the path, we see that the Party utilizes the Ministry of Truth, which is
    related to the Ministry of Peace. The Ministry of Peace, in turn, works for the
    Scientists who are part of the New Aristocracy. This progression reveals a chain of
    command and influence, with different entities working together to serve the Party's
    goals.

Moreover, the inclusion of Journalists in the New Aristocracy indicates the importance of
    media and information dissemination in upholding the Party's ideology. Journalists, as
    part of this elite group, play a role in shaping public opinion and controlling the
    narrative to align with the Party's agenda.

By analyzing these relationships, we can see how the Party exerts control by leveraging
    various sectors of society, such as science, media, and government institutions, to
    maintain its authority, enforce its ideology, and suppress dissent. This interconnected
    web of influence and power dynamics illustrates the extent to which the Party goes to
    ensure its dominance and perpetuate its worldview.

Files Used: {'Full1984.txt'}
```

8 References

1. Athe, P. (2023). Testing Generative Pre-trained Transformer for Knowledge Abstraction and Reasoning for Nuclear Reactor Design and Safety Applications. Retrieved from https://www.ideals.illinois.edu/items/128987.

2. Chen, J., et al. (2023). XplainLLM: Enhancing Model Explainability in NLP. Retrieved from https://arxiv.org/pdf/2311.08614.

3. Ding, L., et al. (2024). Automatic Construction of Theme-specific Knowledge Graphs. Retrieved from https://arxiv.org/pdf/2404.19146.

4. Guidotti, R., et al. (2018). A Survey Of Methods For Explaining Black Box Models. Retrieved from https://arxiv.org/pdf/1802.01933.

5. Lewis, P., et al. (2021). Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. Retrieved from https://arxiv.org/abs/2005.11401.

6. Li, Y., et al. (2024). Building a Knowledge Graph to Enrich ChatGPT Responses in Manufacturing Service Discovery. Retrieved from https://arxiv.org/abs/2404.06571.

7. Lin, L., et al. (2021). Uncertainty Quantification and Software Risk Analysis for Digital Twins in Nearly Autonomous Management and Control Systems: A Review. Retrieved from https://arxiv.org/abs/2103.03680.

8. Stechly, K., et al. (2023). GPT-4 Doesn't Know It's Wrong: An Analysis of Iterative Prompting for Reasoning Problems. Retrieved from https://arxiv.org/pdf/2310.12397.

9. Wang, L., et al. (2022). Development and Application of Data Coverage Assessment for NAMAC Trustworthiness. Retrieved from [Link not available].

10. Wang, L., et al. (2024). Trustworthiness Modeling and Evaluation for a Nearly Autonomous Management and Control Systems. Retrieved from https://www.researchgate.net/publication/378135638_Trustworthiness_modeling_and_evaluation_for_a_nearly_autonomous_management_and_control_system.

11. Xiaorui, S., et al. (2024). Knowledge Graph Based Agent for Complex, Knowledge-Intensive QA in Medicine. Retrieved from https://arxiv.org/pdf/2410.04660.

12. Zirui, G., et al. (2024). LightRAG: Simple and Fast Retrieval-Augmented Generation. Retrieved from https://arxiv.org/abs/2410.05779.