Requirement Discovery Using Embedded Knowledge Graph with ChatGPT

Braxton VanGundy
NASA Langley Research Center
Mail Stop 061
757-776-7247
Braxton.vangundy@nasa.gov

Nipa Phojanamongkolkij
NASA Langley Research Center
Mail Stop 290
757-864-6396
nipa.phojanamongkolkij@nasa.gov

Barclay Brown
Collins Aerospace
400 Collins Rd NE
Cedar Rapids IA 52498, USA
barclay.brown@collins.com

Ramana Polavarapu
NASA Langley Research Center
Mail Stop 050
650-313-4536
ramana.v.polavarapu@nasa.gov

Joshua Bonner
NASA Langley Research Center
Mail Stop 050
678-387-7851
joshua.bonner@nasa.gov

Published by the International Council on Systems Engineering (INCOSE) with permission.

Abstract. The field of Advanced Air Mobility (AAM) is witnessing a transformation with innovations such as electric aircraft and increasingly automated airspace operations. Within AAM, the Urban Air Mobility (UAM) concept focuses on providing air-taxi services in densely populated urban areas. This research introduces the utilization of Large Language Models (LLMs), such as OpenAI's GPT-4, to enhance the UAM Requirement discovery process.

This study explores two distinct approaches to leverage LLMs in the context of UAM Requirement discovery. The first approach evaluates the LLM's ability to provide responses without relying on additional outside systems, such as a relational or graph database. Instead, a vector store provides relevant information to the LLM based on the user's question, a process known as Retrieval Augmented Generation (RAG). The second approach integrates the LLM with a graph database. The LLM acts as an intermediary between the user and the graph database, translating user questions into cypher queries for the database and database responses into human-readable answers for the user. Our team implemented and tested both solutions to analyze requirements within a UAM dataset. This paper will talk about our approaches, implementations, and findings related to both approaches.

Keywords. Large Language Models, OpenAI, ChatGPT, Urban Airspace Mobility, Requirements, Advance Air Mobility, Digital Assistant, Machine Learning, Artificial Intelligence, Graph Database, Link Prediction

Introduction

Advanced Air Mobility encompasses a range of innovative operational and technological changes to aviation (electric aircraft, increasingly automated aircraft, increasingly automated airspace operations, etc.) that are transforming aviation's role in everyday movement of people and goods. The Urban Air Mobility concept covers a subset of the AAM concepts, namely those that provide air-taxi services to the public over densely populated cities and the urban periphery, including flying between local, regional, intra-regional, and urban locations. UAM envisages a future in which advanced technologies and new operational procedures enable practical, cost-effective air transport as an integrated mode of movement of people and goods in metropolitan areas. In this study, UAM operations are limited to those enabled by revolutionary electric Vertical Takeoff and Landing (eVTOL) aircraft designs which are now becoming feasible.

To safely support UAM operations at scale in the National Airspace System (NAS), NASA's Air Traffic Management-Exploration (ATM-X) project is conducting research that evolves the UAM air traffic management system towards a highly automated and operationally flexible system of the future. The complexity of the UAM airspace progression requires a plan to effectively organize, integrate, and communicate NASA's research and development in the area. The UAM airspace system research roadmap, or just roadmap, is a system engineering methodology to manage what is known, what is developed, and what is planned for in NASA's UAM airspace research & development (R&D) lifecycle.

To support the roadmap development lifecycle, the Systems Modeling Language (SysML) standard is used to capture the UAM airspace system decomposition, technical requirements, programmatic task plan, and traceability between the roadmap's technical and programmatic aspects. A thorough description of the roadmap can be found in Levitt et al. (2023) and the detailed description of the Roadmap model can be found in Phojanamongkolkij et al. (2022). All traceability follows the ontology definition given in Figure 1 and can be viewed as the knowledge graph structure.

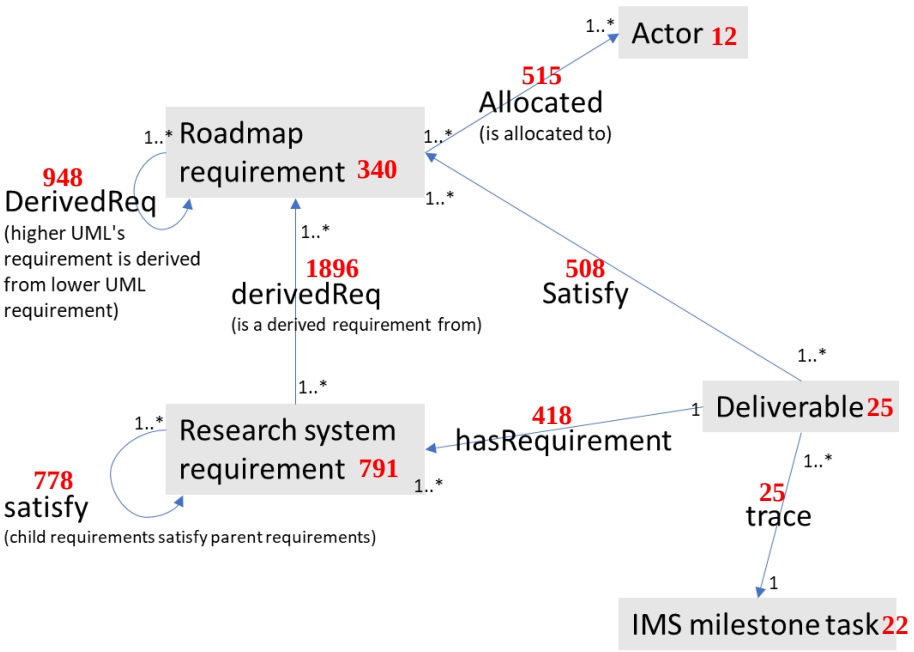

Figure1. Knowledge graph ontology with the red number representing the size of the graph structure.

The ATM-X project has established programmatic milestone tasks in the integrated master schedule (IMS) using MS Project software. These milestones are reported to the project's stakeholders as a means to deliver the project's progress toward stakeholder's expectations. Through ***trace*** relationships, each milestone can be either a package of multiple product deliverables (e.g., software, user manual, technical report) or a single deliverable (e.g., technical report). Whenever the researchers complete the milestone task, its deliverables (e.g., software, user manual, technical report) are reviewed by a systems engineer (SE). The SE qualitatively

assesses which of the roadmap requirements that the deliverables ***satisfy***. One deliverable can satisfy multiple roadmap requirements, and similarly one requirement can be satisfied by multiple deliverables, resulting in a many-to-many satisfy relationship as depicted in the figure above.

Additionally, the SE extracts a set of research system requirements for each deliverable, resulting in one-to-many ***hasRequirement*** relationships. These research system requirements are in fact lower- level requirements (***derivedReq***) from the roadmap requirements. One roadmap requirement can be used to derive multiple research system requirements, and similarly one research system requirement can be derived from multiple roadmap requirements, resulting in a many-to-many relationship. Research system requirements can have multiple levels of specificity. As such, the child requirements ***satisfy*** its parent requirements.

Roadmap requirements describe the future UAM concept of operations (e.g., air taxi) that evolves over time from today's similar operations. At the time of this writing, the element of time evolution is defined by the UAM Maturity Level (UML) varying from UML-2 to UML-4. UML-2 represents initial UAM operations, while UML-4 envisions mature UAM operations. The characteristics of evolving requirements from UML-2 to higher UMLs is defined by ***derivedReq*** relationship. For instance, onboard pilot requirements in UML-2 evolve into remote pilot requirements in UML-4 implying UML-4 requirements are derived from corresponding lower UMLs. One roadmap requirement can be a derived requirement from multiple requirements, resulting in a many-to-many ***derivedReq*** relationship. Finally, roadmap requirements can be ***allocated*** to system actors who are responsible to execute those requirements. One requirement can have multiple responsible actors, and, likewise, one actor is responsible for multiple requirements, representing many-to-many allocated relationships. To explore an innovative digital engineering approach to the established knowledge graph database, the project is beginning to evaluate recent advancements in Large Language Models coupled with the complex graph structure to address systems engineering's concerns about a possibility of missing roadmap requirements. Initial evaluation will limit the edges to the ***allocated*** relationship between requirements and system actors due to two reasons. First, there is no prior existing work undertaking the LLM coupling with graph database and graph data science. Fundamentally, the study aims to explore this approach with a smaller tractable scope instead of the full structure that has 1,190 nodes and 5,088 links, where nodes are requirements, system actors, lower-level requirements, and project deliverables, and links are various relationships between nodes. Second, the commercially available LLM is on the public cloud platform (OpenAI's ChatGPT). In essence, for this exploration the study considers using only the publicly available information where the allocated relationships were implicitly provided in the Levitt et al. (2023).

What are Large Language Models?

Large Language Models are deep neural networks trained to generate text in response to an input of prompt text. LLMs are trained using a very large collection of existing text, in most cases including most information on the Internet from sources such as Wikipedia, books, and web content. In the training process, which for many LLMs can take very large computers several months and which can cost millions of dollars in computing capability and electricity, LLMs attempt to predict words momentarily hidden from sentences in the input training set. By experimentally varying the model's set of training parameters, it "learns" how language is used and can generate language that is likely to appear after a prompt.

Large Language Models are an evolutionary step in the AI specialty of natural language processing, building on earlier concepts including word and sentence vectorization, recurrent neural networks, and long short-term memory networks. To these was added an important new technique known as attention (Vaswani, Shazeer, Parmar, Uszkoreit, Jones, Gomez, ... & Polosukhin, 2017) with the ability to better learn language with words in proper, natural sequence, greatly increasing the naturalness of the language generated.

As LLMs increased in size, they gained new, sometimes unexpected capabilities, including the ability to answer questions of varying levels of complexity, translate from one language to another, and generate software source code in a number of languages based only on their extensive reading of material in that language. Despite their impressive abilities, LLMs are not minds, and do not possess mental states such as intention, belief, desire, or emotion. They make no judgements, nor do they have any views or opinions, even though they may generate language that seems to imply that they do.

What is Knowledge Graph Data Science?

The knowledge graph was introduced into Google's search algorithm in 2012 (Singhal, 2012) to enable its users to discover new, relevant information. Knowledge graphs are based on graph theory, which is a mathematical concept that classifies elements in terms of vertices (nodes) and edges (relationships) to understand connections and patterns within the information being studied.

Several graph database tools were surveyed to determine the best tool suitable for the intended use case of identifying missing requirements. The Neo4j graph database was chosen because it offers a rich Graph Data Science (GDS) library, which includes algorithms apt for our use case. Additionally, Neo4j has generated huge amounts of content online due to its popularity. Since Open AI's ChatGPT pulls data from many online sources, the LLM already has a grasp of Neo4j's query language, called Cypher, eliminating the time to train the LLM for this study.

There are many GDS algorithms readily available in the chosen tool. Readers are referred to the Neo4J graph data science manual (The Neo4J Graph Data Science Library Manual v2.5, n.d.) for thorough descriptions. For this study, the Adamic-Adar Index is used to predict links in a network, i.e., forecasting the likelihood of an edge being created between two nodes. The Adamic-Adar Index is popular for link prediction tasks because of its performance, often outperforming other simpler measures (e.g. counting the number of common neighbors) in predicting future connections; and its simple and intuitive execution: nodes connected by frequently connecting intermediaries are deemed less likely to form future connections.

Exploratory Approaches

Two approaches were explored in this study. For the first approach, the LLM, Open AI's GPT-4, is used to provide a response to a user query without using graph structure. The second approach, the LLM is used as a translation layer between the graph database, Neo4j, and the human user. Details on each approach are given in the following sub-sections. The intent of exploring both approaches is to evaluate whether the LLM has the capability to address this requirement discovery use case on its own or if it requires the integration of an external knowledge management system such as a graph database.

Approach 1: LLM with Retrieval Augmented Generation

The objective of this pure-LLM approach is to determine if an LLM, on its own, without the support of graph database queries, could produce the same kinds of query responses and answers. Seen in figure 2, the first attempted approach used a convention retrieval-augmented generation approach, creating a vector store of the requirements data after splitting the data into chunks or splits, one chunk per requirement. Naturally, this approach worked well for queries which could be answered after the retrieval of a small number of requirements. Since each requirement was several hundred characters in length, an 8k LLM context window could contain only a dozen or so requirements while leaving room for the response and other supplemental prompting.

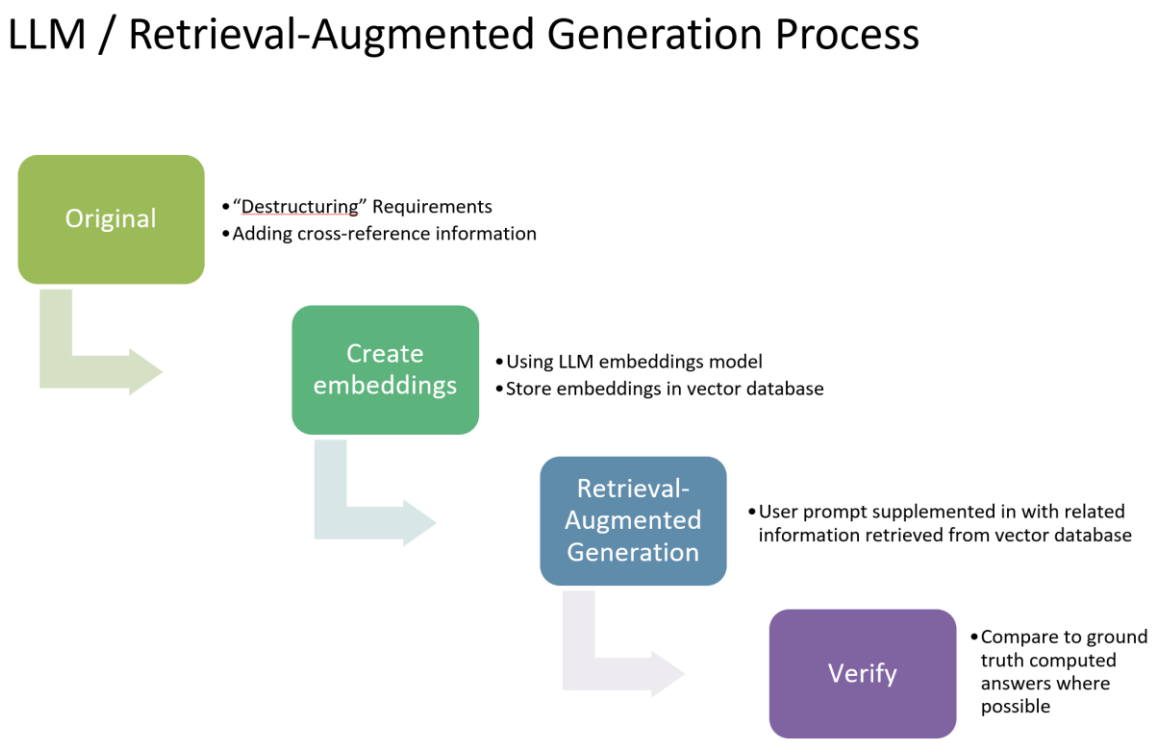

Figure 2. Process for Retrieval Augmented Generation

A query that asked, for example, how many requirements reference a certain keyword or actor is likely to produce an incorrect answer, since the system will retrieve only perhaps ten or so requirements into the context window, based on the vector search. Queries that can be answered by the retrieval of only a few requirements are likely to render correct answers. Given this limitation, a pure RAG approach was discarded as too limiting for the intended queries, many of which were oriented to counting requirements that match a given criterion over a large set of data.

The next approach attempted was to supplement the data stored in the vector database in two primary ways:

1. Convert each requirement and its related data into a single, natural language sentence and store that sentence in the vector database.
2. Create cross reference summaries from the requirements data and including them in the vector database.

Conversation to Natural Language. Each requirement in the original data contains the fields ReqID, Actor, Applied Stereotype (e.g. *allocate*), and Requirement Text. To turn the requirement statement into a more natural language form, the requirement was reformatted as show below. Requirements that were identical except for the actor were combined and the actors made into a comma-separated list.

Requirement <Requirement ID> states <Requirement Text> and has an <Applied Stereotype> relationship connected to the actor(s): <actor list>.

For example:

Requirement ID "19-UML-2.AM" states "When ATC is the designated separator; The Fleet Manager and PIC will receive service from ATC" and has Allocate [Abstraction] relationship connected the actor(s): ATC, Fleet Manager, PIC.

In this way, the structured requirements data became more like natural language narrative, with the intent that an LLM could understand it more successfully. In addition, we expected that the vector similarity function used to retrieve items from the vector store in a RAG approach, would produce better matches when the requirement statement matches more words in the query. For example, a query asking for requirements that

are connected to actor X, is more likely to match the right requirements when the requirement statement itself contains words like “actor” and “connected.”

Cross Reference Items in Vector Store. In early experiments, it was found that when a query required the retrieval of more than just a few requirements, the response would often be incorrect, since the context window would not allow the retrieval of all the necessary requirements. For example, if a query asked for a count of all the requirements connected to a certain actor, the result could well contain more requirements than could be included in the context window, thus the LLM would not be able to successfully count, or answer questions over the entire set.

The technique applied was to add information items to the vector database, which supplied cross reference information across the entire set of requirements. For example, the items of the following structure were generated and added:

> Here is the list of the total of <number of actors>: <actor 1>, <actor 2>… <actor n>.
>
> Here is a list of all of the total of <number of requirements>: <requirement-ID 1>, <requirement-ID 2>, … <requirement-ID n>.
>
> Here is a list of the <relationship> <key, e.g. ReqID> for the <attribute> called <attribute value 1>: <key 1>, <key 2>… <key n>.
>
> Examples:
>
> Here is a list of all of the total of 12 Actors: ATC, UAM Aircraft, PIC, FAA, Vertiport Operator, Fleet Manager, UAM Operator, UAM Community, PSU, PSU Operator, Vertiport Manager, SDSP.
>
> Here is a list of all of the total of 308 ReqIDs: 19-UML-2.AM, 20-UML-2.CS, 21-UML-2.NS, 22-UML-2.SS, 23-UML-2.SS, 25-UML-2.SU, 32-UML-2.AM, 33-UML-3.AM, 34-UML-3.AM, 36-UML-3.AM, 37-UML-4.AM, 38-UML ...
>
> Here is a list of Connected ReqIDs for the Actor called ATC: 19-UML-2.AM, 20-UML-2.CS, 23-UML-2.SS, 53-UML-2.SU, 72-UML-3.AM, 73-UML-3.AM, 89-UML-4.SS, 98-UML-3.AM, 99-UML-3.AM, 119-UML-4.AM, 147-UML-4.AR, 193-UML-2.AD, 194-UML-3.AD, 204-UML-2.SS, 209-UML-3.SS, 222-UML-2.SU, 225-UML-2.SU, 267-UML-2.AD, 275-UML-2.AM, 297-UML-3.AD, 307-UML-3.CS, 333-UML-4.AM, 336-UML-4.CS, 339-UML-4.SS, 428-UML-2.AM, 448-UML-4.CS

These additional items, generated automatically in code and added to the vector store, enable a RAG approach to answer most of the queries that involved counting items. In essence all possible items are pre-counted and listed in separate items, ready for retrieval.

This approach of generating these cross-reference lines should be generalizable to any set of structured data with a few guidelines. In general, if the number of possible values of an attribute in the data is small in relation to the total number of items in the database, then generating a cross-reference item like those above will be effective. Depending on the allowed size of the documents (or splits/chunks) in the vector store, some of the cross-reference lines may need to be split into more than one item. For very large databases with commonly occurring values it is possible that the size of the cross-reference statement could exceed the LLM context window size, in which case that cross reference line could be omitted.

The addition of cross reference lines does not solve all problems with the use of LLMs to query over large datasets. We expect that further evolutions will involve hybrid approaches, including both the generation of structured queries, using graph or SQL approaches, together with the LLM query/prompt capability. The hybrid approach is intuitive and even mirrors how a human would approach a complex query over a large dataset, with the first step being to accumulate all the likely relevant data, then summarizing and reducing this data to what’s needed to answer the question, and then producing a succinct answer. The combination of LLMs and structured queries is likely to produce the best overall outcome.

Approach 2: LLM and Graph Database Integration

A pure RAG approach proved to be inadequate for the UAM Requirement use-case as explored in approach 1. Supplementing the data in the vector store significantly enhanced the accuracy of the LLM's responses, however this approach did not provide the full set of analytical capabilities required to perform analysis on the requirement set. Additionally, our team encountered instances of hallucinations where the LLM would incorrectly count the number of requirements related to a specific system actor with the pure RAG approach and incorrectly map relationships between requirements, even after supplementing the data in the vector store. Within the LLM research space, several approaches have already been attempted to try and mitigate hallucinations by allowing LLMs to consult external resources such as relational databases, document collections, and the broader internet (Andriopoulos & Pouwelse, 2023; Komeili, Shuster, & Weston,2021). The second exploratory approach follows a similar methodology, utilizing a graph database to provide a verified reference for the LLM to pull from. This approach allows us to build a system that makes the advance relationships and algorithms found in today's cutting-edge graph databases available to all users, regardless of their experience with graph databases and the corresponding query language, while ensuring the information provided to the users is correct.

Database Integration. Our architecture uses the LLM as the interface between the user and the graph database. The user asks a question on our web-interface, the LLM converts that question into a cypher query, the database returns a response, and then the LLM provides that response to the user in a human readable format. This process is visualized in the below figure.

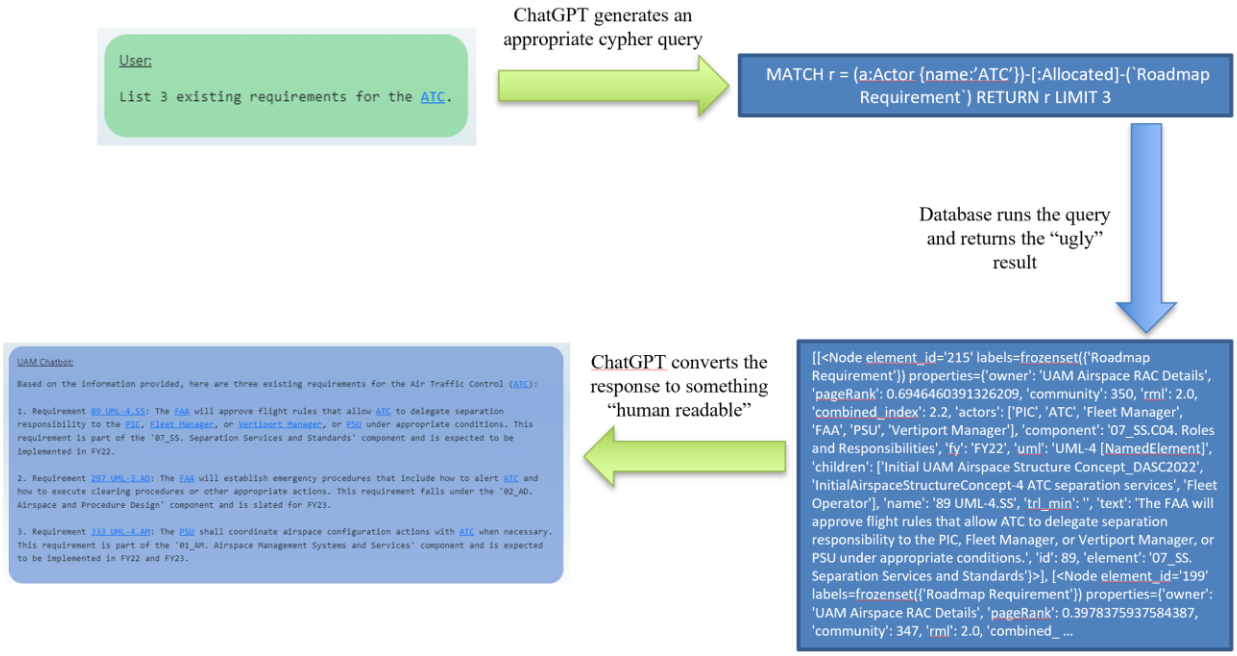

Figure 3. LLM (GPT-4) plus graph database high-level workflow

For each user's question, there are two separate calls to the Azure OpenAI service to get data from the LLM, one to generate the cypher and another to take the response from the database and turn it into human readable text. These calls can be seen shaded in blue in figure 4 below. The first call takes the user's question and converts it into a Neo4j cypher statement. In some cases, specific user questions can cause this part of the process to produce invalid cypher queries so we must programmatically validate the response from the model. This is accomplished by evaluating the structure of the response, if it is representative of a cypher query then the query is sent to the database, otherwise the system will try to generate a response based on the prompt history instead. If the cypher passes the structure check, the database will then run the cypher query and return a large list of information from the graph database. At first glance, most users find the data within this list to be incomprehensible. This is where the second call to the LLM comes into play. The application then takes the difficult-to-read response from the database and sends it back to the model to be converted

into structured English that an average user can read. This is the response that ultimately gets returned to the user in the web-interface.

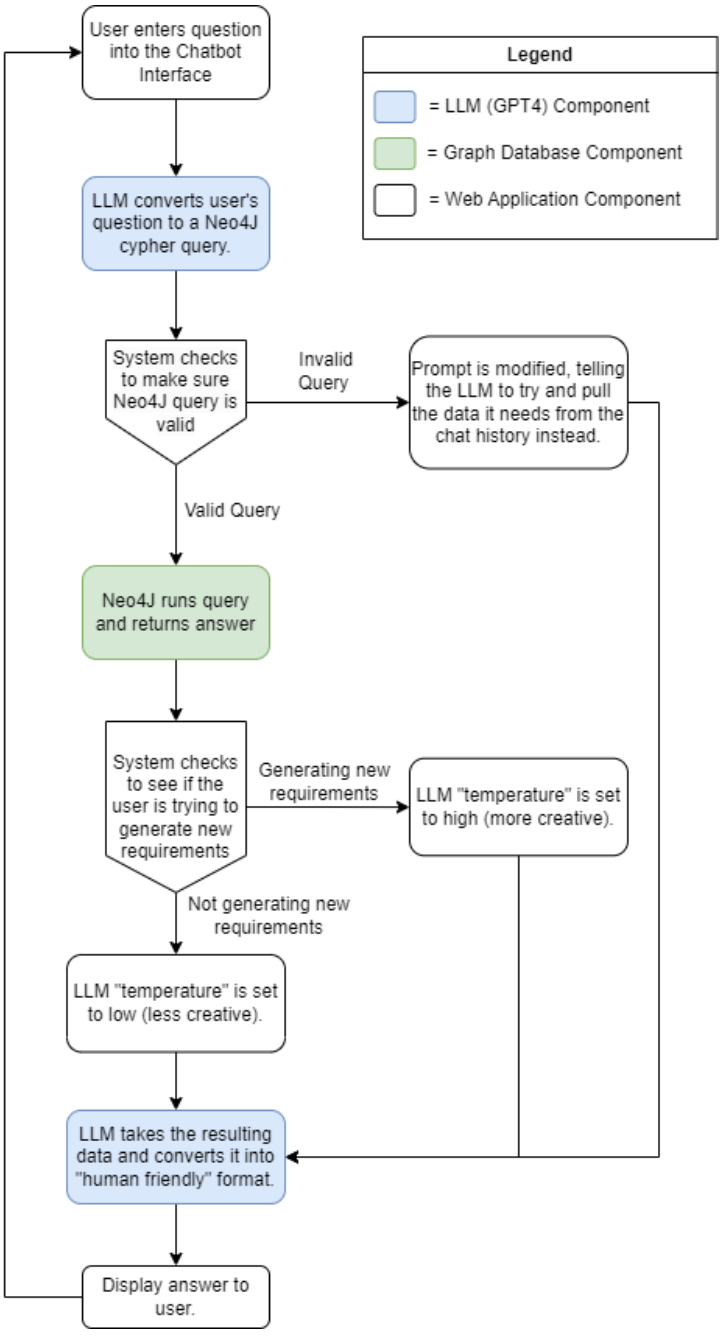

Figure 4. LLM (GPT-4) plus graph database architecture diagram

A notable advantage to using the graph database instead of fine-tuning the LLM directly on the data from the UAM SysML model is the ease of adding new information to the system. If we were to fine-tune the LLM directly on the data from the SysML model, every time we need to add new information, we would have to retrain the model which is a slow, and expensive, operation. With the approach in figure 4, if new information gets added to the SysML model, our team can simply import the new data into the graph database using a few commands and the LLM will immediately have access to it. This also makes this approach easily extensible to other SysML models outside of the UAM research space. New SysML models can be used by setting up the data in a graph database and making slight modifications to the LLM system prompt to account for the new graph database schema.

At a high level this architecture is fairly simple, however there are a few other conditions that need to be accounted for in order to provide some key features within the system. Seen in figure 4, the system can dynamically change the model's "temperature" setting, a value from '0' to '1' which specifies how creative the model should be, depending on the user's question. In most cases when working with the graph database, we want this temperature setting to be '0' which is the least creative setting. Setting the temperature to '0' helps ensure GPT-4 returns the information exactly as provided by the graph database. However, there are some cases where we do want the model to exhibit creativity, specifically when generating descriptions for new requirements. A key functionality of our system is to be able to take a set of predicted requirements, found using the graph database's link prediction capability, and use those requirements to generate possible descriptions for new requirements that have never been seen before. In this case, if the system detects that the user is asking to generate a set of new requirements, it will increase the temperature to '0.6'.

Prompt Engineering. As mentioned previously, our approach makes two calls to the Azure OpenAI GPT-4 service every time a user asks a question. Each of these calls utilizes its own unique prompt. The first prompt takes the user's question and converts it into a cypher query that is readable by the Neo4j graph database. The second prompt takes the response from the Neo4J graph database, formatted as a large list, and converts it into structured English. Note that figure 5 and figure 6 provided below are high level depictions of the prompts we used and do not contain the full text prompts we are currently using in production.

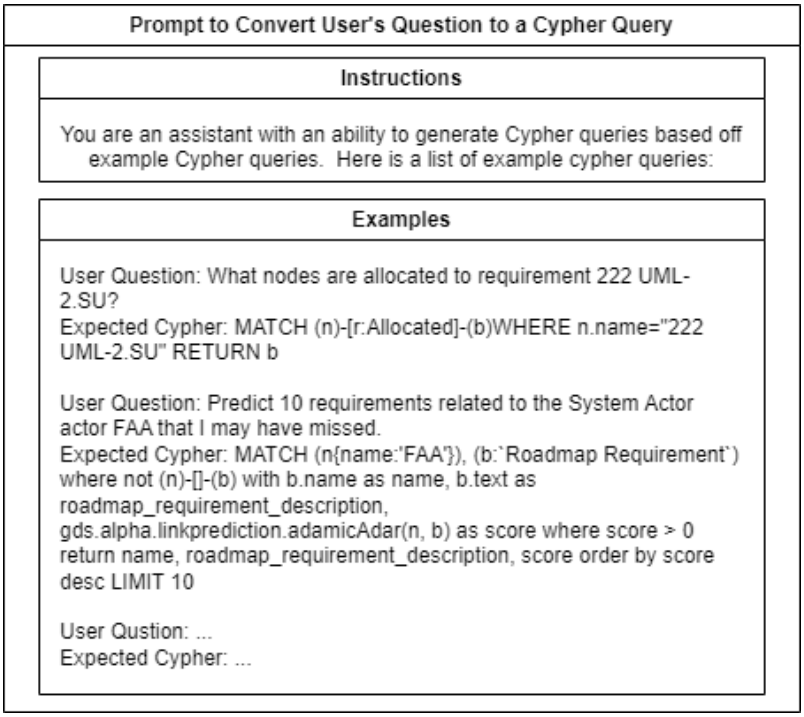

Figure 5. Prompt to convert questions to cypher

Starting with the prompt where the user's question gets converted to a cypher query, the model is first provided with a role as seen in the "Instruction" section in figure 5. In this case, the role of GPT-4 is to convert the user's question into an equivalent database query. The LLM is provided 19 unique examples as a reference, the general structure for those examples can be seen in the "Examples" section in figure 5. Each example contains a potential user question along with the expected query to be returned from the model. It is not necessary to provide every possible user question / cypher query permutation. In fact, this is one of the reasons that we specifically chose Neo4J as our graph database. Thanks to Neo4j's popularity, there are a plentiful number of cypher query examples scattered throughout the internet, many of those appear to have been incorporated into the GPT-4 training data. With only 19 provided examples, determined by our team to be the minimum number of examples required to generate reliable cypher queries, the model can restructure the example queries to work for hundreds of potential user questions. Some queries, such as those required for link prediction, are significantly complex, spanning over 400 characters, containing several different conditions, and requiring calls to the graph data science library. Following some light prompt engineering by our

team, it was found that the LLM could consistently produce these complex queries, such as the second example in figure 5.

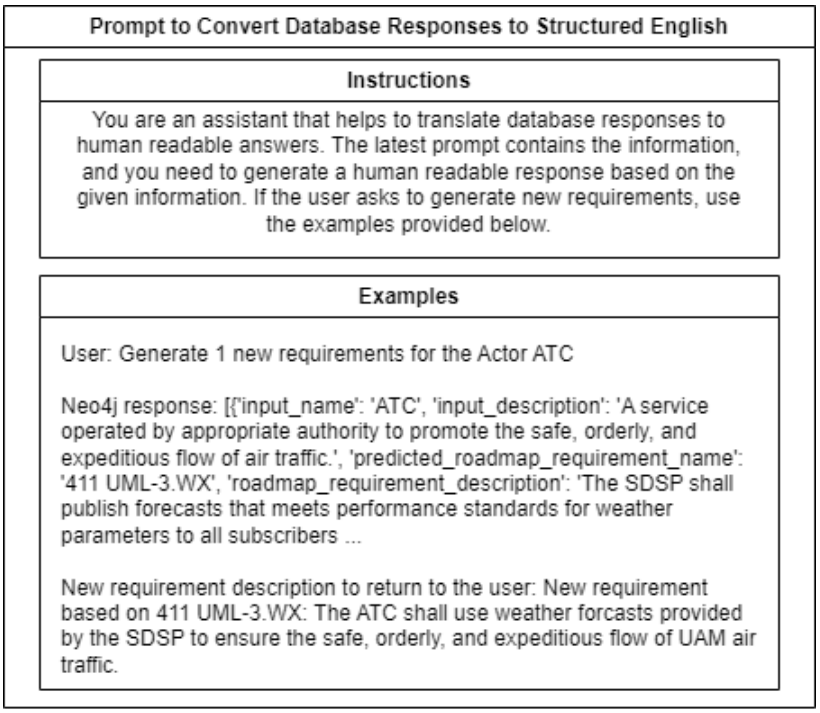

Figure 6. Prompt to convert database response to human readable text and create a new requirement from a predicted requirement

Assuming the cypher generated by the first call to the LLM is valid, the database will return the requested information in a JSON format. The next step is to take this JSON and generate a human readable response from it by making a second call to the LLM. In many cases this human readable response may look like a numbered list or a couple sentences describing a particular requirement or relationship. Our team found that it took only a few sentences to instruct the model on how to convert the JSON to the desired structured English format, GPT-4 appears to be well adapted to converting structured data to plain English. As seen in figure 6, once again, the prompt starts by defining a role for the model in the “Instructions” section of the diagram. However, the examples provided this time are not depicting English to cypher translations, instead we are telling the system what to do if the user asks to generate new requirements. This is a key feature of our UAM Chatbot, UAM researchers want to be able to take the predicted requirements found by the link prediction algorithms within Neo4j and combine them with the source requirement to generate new requirement sets. We found that GPT-4 did not know how to do that out of the box, so we provided examples as seen above in figure 6.

Implementation and Design.

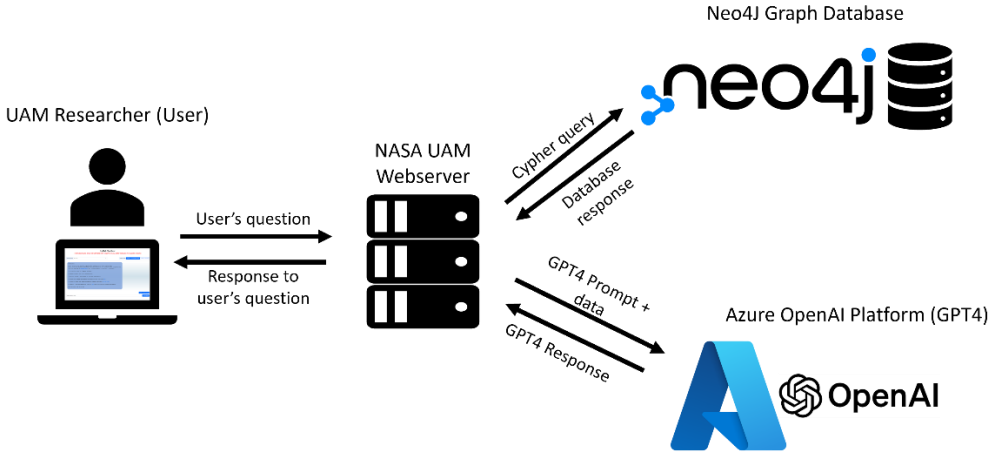

Figure 7. High level application architecture

Our application is comprised of three separate systems working together to generate answers for UAM Researchers. At the center of figure 7 is the webserver that serves the user interface to the user and functions as an intermediary between our Neo4j graph database, the LLM, and the user. The graph database has been curated by our UAM Research team to ensure the data is accurate and the responses from the LLM have been tested to ensure the answers are represented exactly as they are in the graph database.

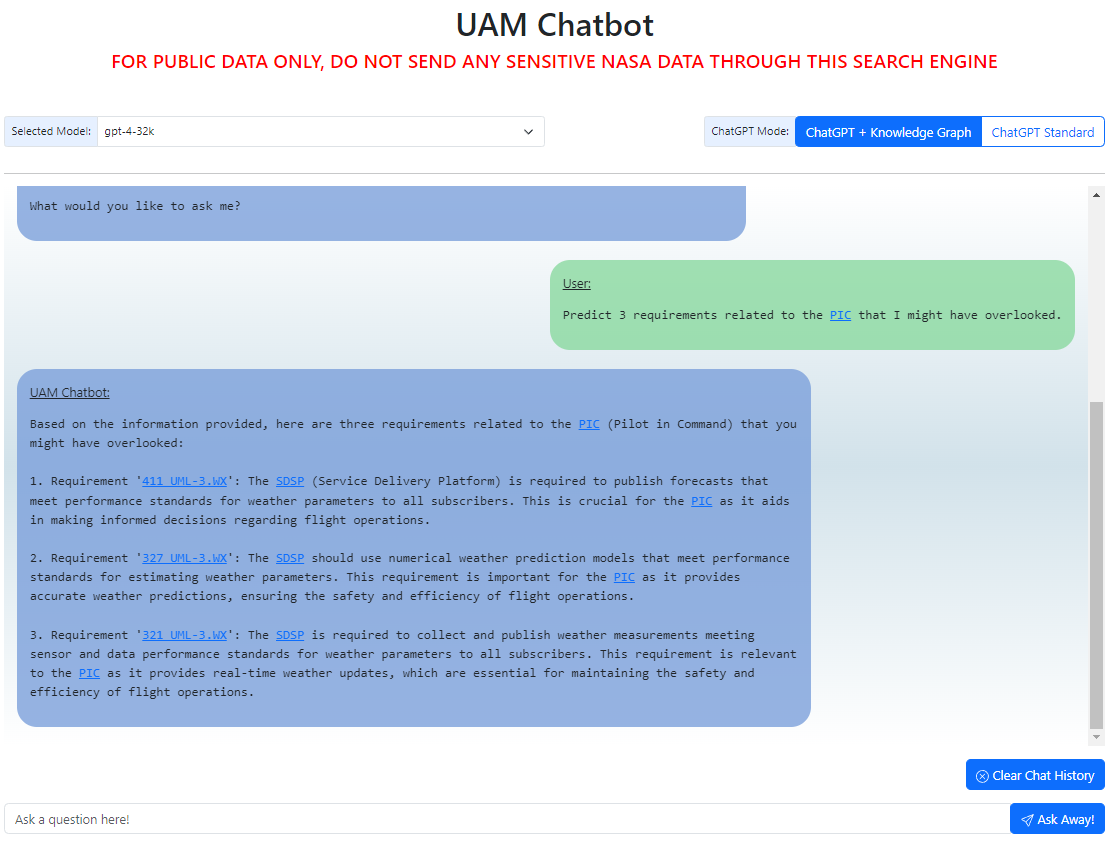

Figure 8. UAM application interface

The user interface in figure 8 is modeled off typical Short Message Service (SMS) messaging applications. The user's question gets placed on the right side of the window and the chatbot's responses are placed to the left. The colors are similar to those used in the iOS messaging application, the color green is used for sent "messages" and blue is used to indicate responses.

Approach Comparison

Both approaches were able to generate answers for questions regarding UAM Requirements. However, approach 1, LLM with RAG (Azure OpenAI GPT-4), was more adept at answering generalized questions and fell short when it came to numerical analysis, finding existing links, or questions that required advanced graph analysis techniques such as requirement generation using link prediction. Seen in table 1, when asked to find requirements related to UAM Aircraft, approach 1 identified only four of the six requirements that should have been associated with that system actor. This approach was also incapable of predicting requirements and, because of this, was not able to generate new requirements based off predicted requirements. In many use-cases this may be acceptable, especially if no existing relational or graph databases are available. Though, the conclusions from such an approach are difficult to validate since they are based around an NLP analysis of each requirement's description instead of a structured graph ontology. Our team had to look at each requirement's description to determine why the LLM might have come to a specific conclusion, and in some cases, it was not entirely clear. Approach 2, Graph Database / Data Science + LLM (Azure OpenAI GPT-4), removes much of this uncertainty from the equation by utilizing known link prediction algorithms to perform analysis on the requirements. Additionally, the graph database handles all numerical analysis, and all existing nodes and edges in the database have been verified by our researchers. Approach 2, while requiring additional setup, cost, and ongoing maintenance, as seen in table 2, proved to be the better solution for our specific use-case. Approach 2 was able to answer all user questions provided in table 1. However, when asked to provide a summary of the responsibilities for a specific system actor, the user was required to first ask for the requirements related to that system actor and then a second question was sent to the system asking to condense the related requirements down into responsibilities. It should be noted that our set of questions targeting UAM requirements are more suited for graph queries, so it is not surprising to see that approach 2 is a better fit for this application. If this use-case was more centered around summarizing content or deriving meaning from requirements, then approach 1 might have been the better fit. Going forward it may be possible to combine these approaches into a hybrid system to allow the user to choose between using a vector store for data that is not stored in an existing database or allowing them to use the LLM in combination with an existing database to perform analysis on their requirements.

Table 1. Approach response comparisons

User Question	Approach 1: LLM with RAG (Azure OpenAI GPT-4)	Approach 2: Graph Database / Data Science + LLM (Azure OpenAI GPT-4)
How many unique requirements do we have for the System?	Capable of answering	Capable of answering
How many unique system actors do we have?	Capable of answering	Capable of answering
Please list the unique system actors	Capable of answering	Capable of answering
Give me 5 requirements with multiple actors.	Capable of answering	Capable of answering
Find all requirements where one actor is UAM Aircraft	Only Finds 4 of 6	Capable of answering
Predict five requirements for UAM Aircraft that I may have missed	Not able to answer	Capable of answering
Create five new requirements for UAM Aircraft	Capable of answering	Capable of answering
Summarize the responsibilities of the UAM Operator	Capable of answering	Capable of answering, requires two separate questions from the user.

Color Key: Acceptable Answer - | Partially Acceptable Answer - | Unacceptable Answer -

Table 2. Approach resource Comparisons

Aspects	Approach 1: LLM with RAG (Azure OpenAI GPT-4)	Approach 2: Graph Database / Data Science + LLM (Azure OpenAI GPT-4)
Implementation	Moderate: Retrieval-augmented generation pattern using LLM and vector store	Difficult and more expensive, requires deployment of supporting infrastructure and curation of graph database.
Cost for cloud server and developer	• OpenAI subscription	• Open AI subscription • Server for graph database • Server for web-application • Application / system administrator
Quality of solutions	Varies based on type of query and structure of embedding vector store: better on general open queries, worse on numerical counting	Using the graph database provides a ground truth, chances of hallucination are much lower (almost 0 but not quite since GPT-4 is a "black-box"). Capable of providing accurate answers for questions involving numerical counting. Provides advance graph algorithms to the user.
Lessons Learned	• Controlled Unclassified Information (CUI) not permitted	• CUI not permitted • Difficult to get consistent, complex, cypher queries from GPT. Took many instances of trial and error to come up with a functional prompt.

Conclusion and Next Steps

Ultimately, our team decided to deploy the LLM and Graph Database Integration approach (approach 2) for our production UAM Requirement system. This approach allows our SEs to utilize the powerful graph

database algorithms found within Neo4J, giving them the ability to analyze connections between existing requirements and actors, predict new connections, and generate new requirements that are comparable to those created by our subject matter experts. By using the LLM as a translational layer, our users are not required to possess existing knowledge of how to formulate graph database queries, and since the information within the database has been curated by UAM Researchers, the chances of the system hallucinating information are greatly reduced. Our System Engineers have reported that this system has increased their speed of requirement analysis by 7x, and that requirements generated by this system are comparable to those created by the human users.

Moving forward, our team plans to expand the scope of our data beyond system actors and roadmap requirements to the full structure of our knowledge graph ontology. This will entail adding in research system requirements, deliverables, IMS milestones, and their respective relationships to encompass all 1,190 nodes and 5,088 links. It may also be possible to apply a hybrid approach to our requirements system, utilizing a combination of graph databases, relational databases, and vector stores, and allowing the LLM to choose the source to query from based on the user's question. Furthermore, our team is evaluating the feasibility of deploying other open-source models within our organization to allow our system to be applied to use-cases beyond UAM Research that may contain sensitive information.

References

Andriopoulos, K., & Pouwelse, J. (2023). Augmenting LLMs with Knowledge: A survey on hallucination prevention. arXiv preprint arXiv:2309.16459.

Komeili, M., Shuster, K., & Weston, J. (2021). Internet-augmented dialogue generation. arXiv preprint arXiv:2107.07566.

Levitt, I., Phojanamongkolkij, N., Horn, A., & Witzberger, K. (2023). UAM Airspace Research Roadmap-Rev. 2.0.

Phojanamongkolkij, N., Levitt, I. M., & Barnes, P. D. Overview of Model-Based Systems Engineering Efforts to Evolve the Airspace Research Roadmap. In AIAA AVIATION 2022.

Singhal, A. (2012, May 16). Introducing the Knowledge Graph: things, not strings. Google. https://blog.google/products/search/introducing-knowledge-graph-things-not/

The Neo4J Graph Data Science Library Manual v2.5 - NEO4J Graph Data Science. (n.d.). Neo4j Graph Data Platform. https://neo4j.com/docs/graph-data-science/current

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. Advances in neural information processing systems, 30.

Biography

Braxton VanGundy is a Software Engineer for the Flight Software Systems branch at NASA Langley Research center. He holds a bachelor's in Computer Engineering and a minor in Computer Science from Old Dominion University (ODU) in addition to a master's in Engineering Management from ODU. He has been at NASA Langley since 2017 working on various projects related to data science, big data, and flight software.

Nipa Phojanamongkolkij is the Systems Engineer for Air Traffic Management Exploration project at NASA. She holds both a Master and Doctorate degrees in industrial and systems engineering from Arizona State University. She collaborates with project managers, engineers, and scientists through working groups, technical forums, and has authored several Model-based Systems Engineering and Digital Engineering technical papers. She served as a statistical engineer and systems engineer in science and aeronautics research projects.

Barclay Brown is Associate Director for Research in AI at Collins Aerospace, a division of RTX. Before joining Collins, he was an Engineering Fellow in Raytheon Missiles and Defense, focusing on AI and systems engineering methods, and prior to that he was the Global Solution Executive for the Aerospace and Defense Industry at IBM. He is author of Engineering Intelligent Systems, a certified expert systems engineering professional (ESEP), former CIO of INCOSE, and chair of the INCOSE AI Systems working group.

Ramana Polavarapu is a senior data scientist at NASA. He secured a PhD in Economics from The University of California, Davis. Dr. Polavarapu was previously employed as a vice president and a tech fellow at Goldman Sachs, research scientist at Amazon, and an ML engineer and Research Scientist at Facebook (Meta).

Joshua Bonner is a data scientist for the Office of the Chief Information Officer at NASA. He previously contributed to the covid response as a health statistician and data scientist for the Centers of Disease Control and Prevention; was employed as a biostatistician at Walter Reed National Naval Medical Center; and regularly collaborated with the NIH as a statistical consultant.