How to Convert Any Text Into a Graph of Concepts

A method to convert any text corpus into a Knowledge Graph using Mistral 7B

Rahul Nayak · Follow
Published in Towards Data Science
12 min read · Nov 10, 2023

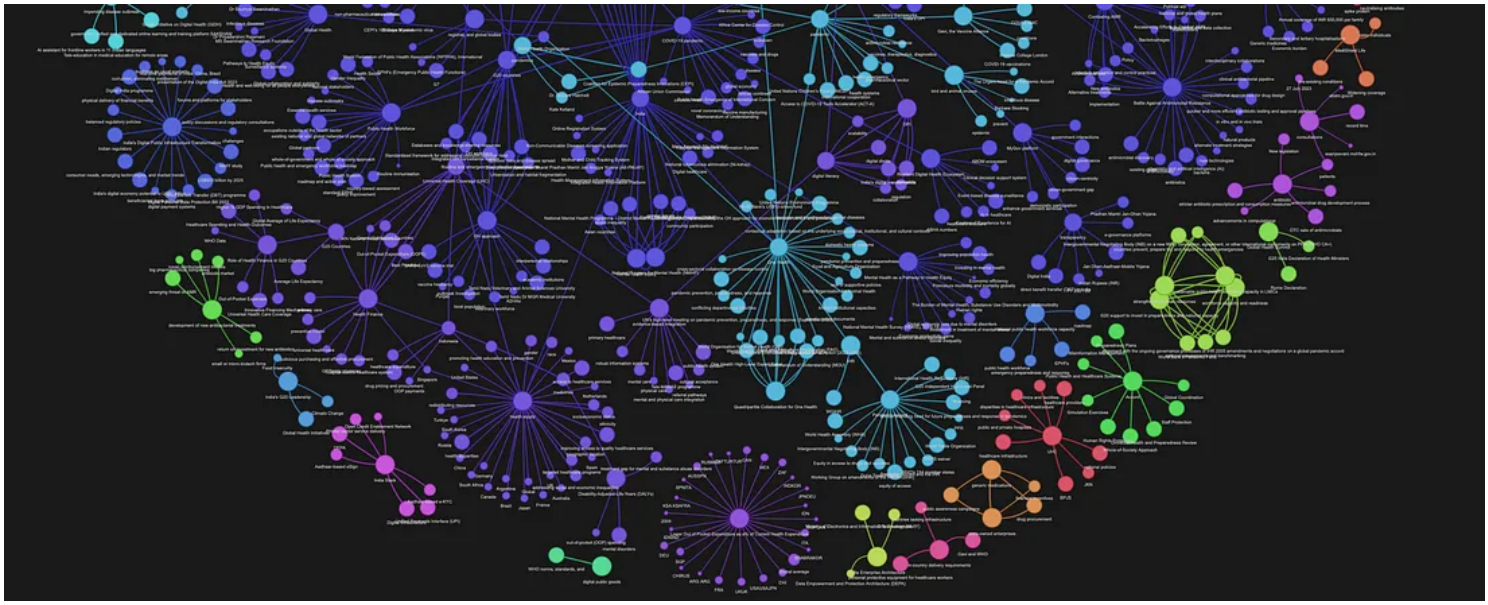


Image generated by the author using the project shared in this article.

A few months ago, knowledge-based QnA (KBQA) was a novelty. Now KBQA with Retrieval Augmented Generation (RAG) is a piece of cake for any AI enthusiast. It's fascinating to see how the realm of possibilities in NLP has expanded so rapidly due to LLMs. And it's getting better by the day.

In my last article, I shared a recursive RAG approach to implement QnA with multi-hop reasoning to answer complex queries based on a large corpus of text.

The Research Agent: Addressing the Challenge of Answering Questions Based on a Large Text Corpus

I made an Autonomous AI Research Agent that can answer difficult questions with deep multi-hop reasoning capabilities

towardsdatascience.com

A good number of folks tried it out and sent their feedback. Thanks all for your feedback. I have since collated these contributions and made a few improvements to the code to address some of the problems with the original implementation. I plan to write a separate article about it.

In this article, I want to share another idea that may help create a super research agent when combined with recursive RAG. The idea emerged out of my experiments with recursive RAG with smaller LLMs, and a few other ideas that I read on Medium — specifically one, the **Knowledge-Graph Augmented Generation.**

Abstract

A Knowledge Graphs (KG), or any Graph, is made up of Nodes and Edges. Each node of the KG represents a concept and each edge is a relationship between a pair of such concepts. In this article, I will share a method to convert any text corpus into a Graph of Concepts. I am using the term 'Graph of Concept' (GC) interchangeably with the terms KG to better describe what I am demoing here.

All the components I used in this implementation can be set up locally, so this project can be run easily on a personal machine. I have adopted a no-GPT approach here because I believe in smaller open source models. I am using the fantastic Mistral 7B Openorca instruct and Zephyr models. These models can be set up locally with Ollama.

Databases like Neo4j make it easy to store and retrieve graph data. Here I am using in-memory Pandas Dataframes and the NetworkX Python library, to keep things simple.

Our goal here is to convert any text corpus into a Graph of Concepts (GC) and visualise it like the beautiful banner image of this article. We will even interact with the network

graph by moving nodes and edges, zooming in and out, and change the physics of the graph to our heart's desire. Here is the Github page link that shows the result of what we are building.

https://rahulnyk.github.io/knowledge_graph/

But first, let's delve into the fundamental idea of KGs and why we need them. If you are familiar with this concept already, feel free to skip the next section.

Knowledge Graph

Consider the following text.

Mary had a little lamb,
You've heard this tale before;
But did you know she passed her plate,
And had a little more!

(I hope the kids are not reading this 😁)

Here is one possible representation of the text as a KG.

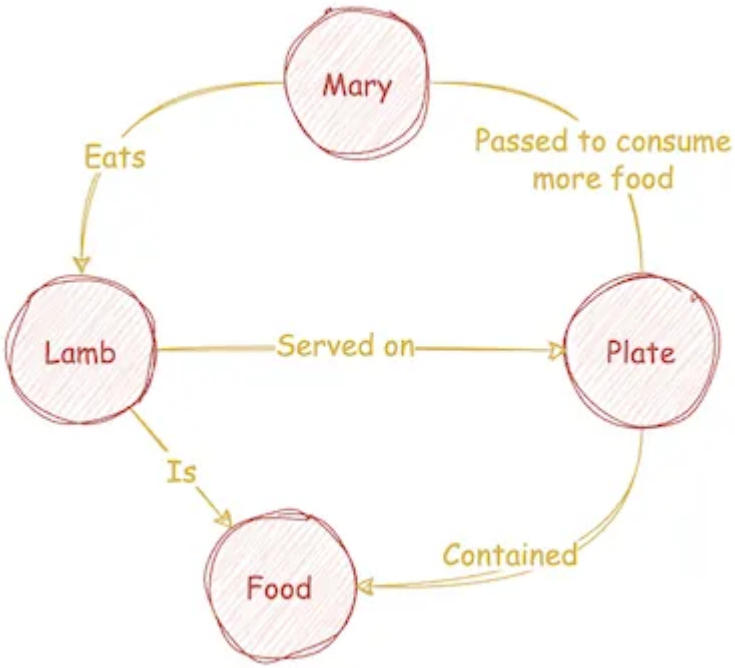


Diagram created by the Author using draw.io

The following article from IBM aptly explains the fundamental concept of Knowledge Graphs.

What is a Knowledge Graph? | IBM

Learn about knowledge graphs, networks of semantic metadata which represent a collection of related entities.

www.ibm.com

Quoting an excerpt from the article to summarise the idea:

> *A knowledge graph, also known as a semantic network, represents a network of real-world entities — i.e. objects, events, situations, or concepts — and illustrates the relationship between them. This information is usually stored in a graph database and visualised as a graph structure, prompting the term knowledge "graph."*

Why Knowledge Graph?

Knowledge Graphs are useful in a variety of ways. We can run graph algorithms and calculate the centralities for any node, to understand how important a concept (node) is to the body of work. We can analyse connected and disconnected sets of concepts, or

calculate communities of concepts for a deep understanding of the subject matter. We can understand links between seemingly disconnected concepts.

We can also use knowledge graphs to implement Graph Retrieval Augmented Generation (GRAG or GAG) and chat with our documents. This can give us much better results than the plain old version of RAG, which suffers several shortcomings. For example, retrieving the context that is the most relevant for the query with a simple semantic similarity search is not always effective. Especially, when the query does not provide enough context about its true intent, or when the context is fragments across a large corpus of text.

For example, consider this query —

Tell me about the family tree of José Arcadio Buendía in the book 'One Hundred years of Solitude'.

The book documents 7 generations of *José Arcadio Buendía* with half of the characters named *José Arcadio Buendía*. It will be quite a challenge, if even possible, to answer the query using a simple RAG pipeline.

Another shortcoming of RAG is that it can't tell you what to ask. Very often, asking the

Retrieval Augmented Generation pipeline to get the best of both worlds.

So now we know that Graphs are interesting, they can be extremely useful, and they also look beautiful.

Creating the Graph of Concepts

If you were to ask GPT, how to create a graph of knowledge from a given text? it may suggest a process like the following.

1. Extract concepts and entities from the body of work. These are the nodes.
2. Extract relations between the concepts. These are the edges.
3. Populate nodes (concepts) and edges (relations) in a graph data structure or a graph database.
4. Visualise, for some artistic gratification if nothing else.

Steps 3 and 4 sound understandable. But how do you achieve steps 1 and 2?

Here is a flow diagram of the method I devised to extract a graph of concepts from any given text corpus. It is similar to the above method but for a few minor differences.

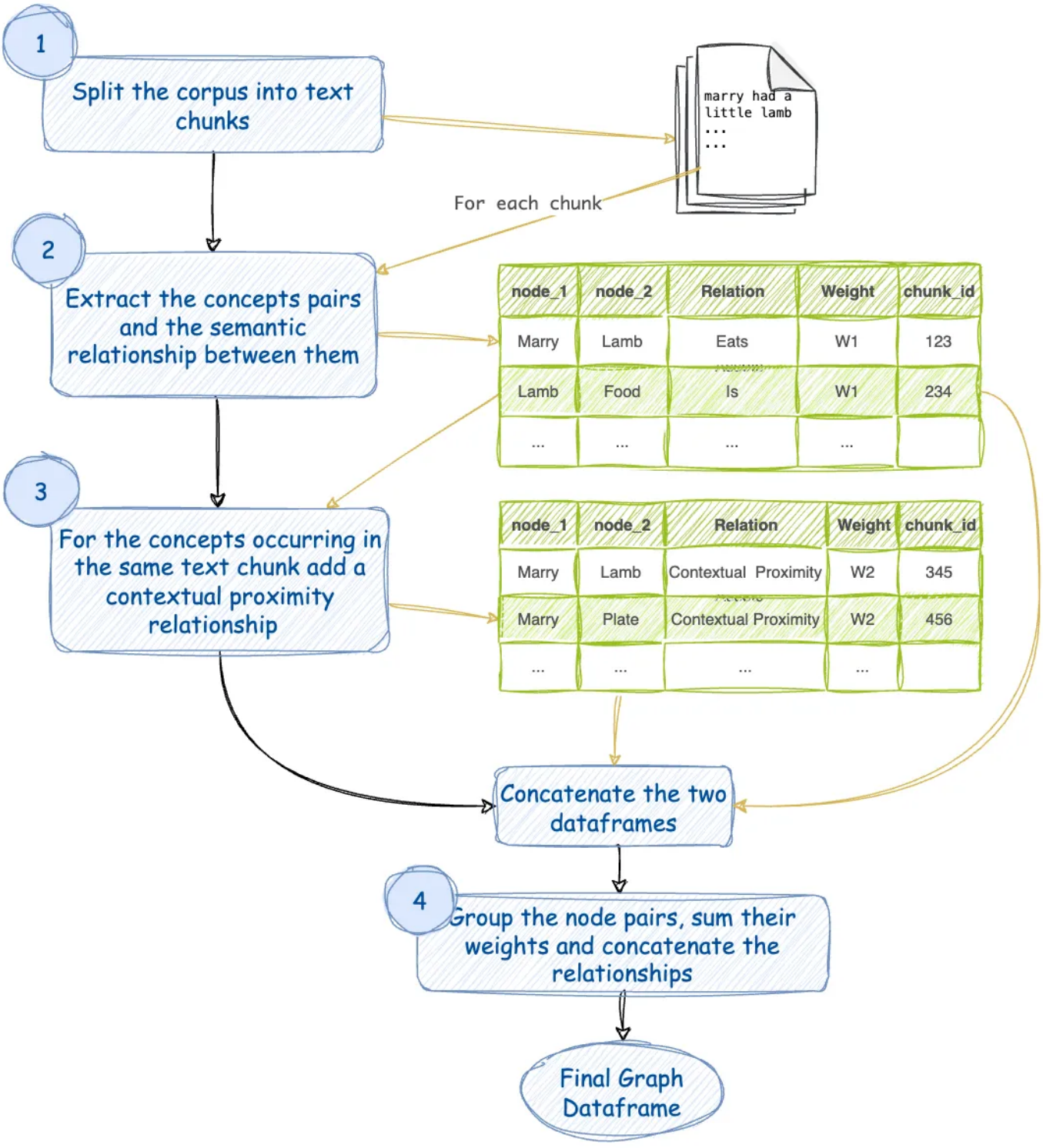


Diagram created by the Author using draw.io

1. Split the corpus of text into chunks. Assign a chunk_id to each of these chunks.

2. For every text chunk extract concepts and their semantic relationships using an LLM. Let's assign this relation a weightage of W1. There can be multiple relationships between the same pair of concepts. Every such relation is an edge between a pair of concepts.

3. Consider that the concepts that occur in the same text chunk are also related by their contextual proximity. Let's assign this relation a weightage of W2. Note that the same pair of concepts may occur in multiple chunks.

4. Group similar pairs, sum their weights, and concatenate their relationships. So now we have only one edge between any distinct pair of concepts. The edge has a certain weight and a list of relations as its name.

You can see the implementation of this method as a Python code, in the GitHub repository I share in this article. Let us briefly walk through the key ideas of the implementation in the next few sections.

To demonstrate the method here, I am using the following review article published in PubMed/Cureus under the terms of the Creative Commons Attribution License. Credit to the authors at the end of this article.

India's Opportunity to Address Human Resource Challenges in Healthcare

India's health indicators have improved in recent times but continue to lag behind those of its peer nations. The...

www.cureus.com

The Mistral and the Prompt

Step 1 in the above flow chart is easy. Langchain provides a plethora of text splitters we can use to split our text into chunks.

Step 2 is where the real fun starts. To extract the concepts and their relationships, I am using the Mistral 7B model. Before converging on the variant of the model best suited for our purpose, I experimented with the following:

Mistral Instruct
Mistral OpenOrca, and
Zephyr (Hugging Face version derived from Mistral)

I used the 4-bit quantised version of these models — So that my Mac doesn't start hating me — hosted locally with Ollama.

Ollama

Get up and running with large language models, locally.

ollama.ai

These models are all instruction-tuned models with a system prompt and a user prompt. They all do a pretty good job at following the instructions and formatting the answer neatly in JSONs if we tell them to.

After a few rounds of hit and trial, I finally converged on to the **Zephyr model** with the following prompts.

```
SYS_PROMPT = (
    "You are a network graph maker who extracts terms and their relations from a g
    "You are provided with a context chunk (delimited by ```) Your task is to extr
    "of terms mentioned in the given context. These terms should represent the key
    "Thought 1: While traversing through each sentence, Think about the key terms
        "\tTerms may include object, entity, location, organization, person, \n"
        "\tcondition, acronym, documents, service, concept, etc.\n"
        "\tTerms should be as atomistic as possible\n\n"
    "Thought 2: Think about how these terms can have one on one relation with othe
        "\tTerms that are mentioned in the same sentence or the same paragraph are
        "\tTerms can be related to many other terms\n\n"
    "Thought 3: Find out the relation between each such related pair of terms. \n"
    "Format your output as a list of json. Each element of the list contains a pa
    "and the relation between them, like the follwing: \n"
    "[\n"
    "   {\n"
    '       "node_1": "A concept from extracted ontology",\n'
    '       "node_2": "A related concept from extracted ontology",\n'
    '       "edge": "relationship between the two concepts, node_1 and node_2 in c
    "   }, {...}\n"
    "]"
)

USER_PROMPT = f"context: ```{input}``` \n\n output: "
```

If we pass our (not fit for) nursery rhyme with this prompt, here is the result.

```
[
  {
    "node_1": "Mary",
    "node_2": "lamb",
    "edge": "owned by"
  },
  {
    "node_1": "plate",
    "node_2": "food",
    "edge": "contained"
  }, . . .
]
```

Notice, that it even guessed ‘food’ as a concept, which was not explicitly mentioned in the text chunk. Isn’t this wonderful!

If we run this through every text chunk of our example article and convert the json into a Pandas data frame, here is what it looks like.

	node_1	node_2	edge	chunk_id	count
0	india's health indicators	peer nations	continue to lag behind	ae0fd26675d645e787964255667e90f4	4
2	health workers density	doctors and nurses/midwives	for 10,00 persons	ae0fd26675d645e787964255667e90f4	4
4	skilled health workforce	india	reinforces the central role human resources ha...	ae0fd26675d645e787964255667e90f4	4
5	skewed inter-state	urban-rural	and public-private sector divide	ae0fd26675d645e787964255667e90f4	4
7	health budget	federal	offers an unprecedented opportunity to do this	ae0fd26675d645e787964255667e90f4	4

Every row here represents a relation between a pair of concepts. Each row is an edge between two nodes in our graph, and there can be multiple edges or relationships between the same pair of concepts. The count in the above data frame is the weight that I arbitrarily set to 4.

Contextual Proximity

I assume that the concepts that occur close to each other in the text corpus are related. Let’s call this relation ‘contextual proximity’.

To calculate the contextual proximity edges, we melt the dataframe so that node_1 and node_2 collapse into a single column. Then we create a self-join of this dataframe

using the chunk_id as the key. So nodes that have the same chunk_id will pair with each other to form a row.

But this also means that each concept will also be paired with itself. This is called a self-loop, where an edge starts and ends on the same node. To remove these self-loops, we will drop every row where node_1 is the same as node_2 from the dataframe.

In the end, we get a dataframe very similar to our original dataframe.

	node_1	node_2	chunk_id	count	edge
2827	world-class health facilities	nhm strategies	0857ab4513ad4383aed095bcf24506fa,0857ab4513ad4...	10	contextual proximity
2828	world-class health facilities	rural areas	0857ab4513ad4383aed095bcf24506fa,0857ab4513ad4...	2	contextual proximity
2829	world-class health facilities	social norms	0857ab4513ad4383aed095bcf24506fa,0857ab4513ad4...	2	contextual proximity
2830	world-class health facilities	urban areas	0857ab4513ad4383aed095bcf24506fa,0857ab4513ad4...	2	contextual proximity
2831	world-class health facilities	urban slums	0857ab4513ad4383aed095bcf24506fa,0857ab4513ad4...	2	contextual proximity

The count column here is the number of chunks where node_1 and node_2 occur together. The column chunk_id is a list of all these chunks.

So we now have two dataframes, one with the semantic relation, and another with the contextual proximity relation between concepts mentioned in the text. We can combine them to form our network graph dataframe.

We are done building a graph of concepts for our text. But to leave it at this point will be quite an ungratifying exercise. Our goal is to visualise the Graph just like the featured image at the beginning of this article, and we are not far from our goal.

Creating a Network of Concepts

NetworkX is a Python library that makes dealing with graphs super easy. If you are not already familiar with the library, click their logo below to learn more

NetworkX - NetworkX documentation

NetworkX is a Python package for the creation, manipulation, and study of the structure, dynamics, and functions of...

networkx.org

Adding our dataframe to a NetworkX graph is just a few lines of code.

```
G = nx.Graph()

## Add nodes to the graph
for node in nodes:
    G.add_node(str(node))

## Add edges to the graph
for index, row in dfg.iterrows():
    G.add_edge(
        str(row["node_1"]),
        str(row["node_2"]),
        title=row["edge"],
        weight=row['count']
    )
```

This is where we can start harnessing the power of Network Graph. NetworkX provides a plethora of network algorithms out of the box for us to use. Here is a link to the list of algorithms we can run on our Graph.

Algorithms - NetworkX 3.2.1 documentation

Edit description

networkx.org

Here, I use a community detection algorithm to add colours to the nodes. Communities are groups of nodes that are more tightly connected with each other, than with the rest of the graph. Communities of concepts can give us a good idea of broad themes discussed in the text.

The Girvan Newman algorithm detected 17 communities of concept in the Review Article we are working with. Here is one such community.

```
[
  'digital technology',
  'EVIN',
  'medical devices',
```

```
    'online training management information systems',
    'wearable, trackable technology'
]
```

This immediately gives us an idea about the broad theme of health technologies discussed in the review paper and enables us to ask questions that we can then answer with our RAG Pipeline. Isn't that great?

Let us also calculate the degree of each concept in our graph. The degree of a node is the total number of edges it is connected with. So in our case, the higher the degree of a concept, the more central it is to the subject of our text. We will use the degree as the size of the node in our visualisation.

Graph Visualisation

Visualisation is the most fun part of this exercise. It has a certain quality to it that gives you an artistic gratification.

I am using the PiVis library to create interactive graphs. Pyvis is a Python library for visualizing networks. Here is a medium article that demonstrates the ease and the power of the Library

Pyvis: Visualize Interactive Network Graphs in Python

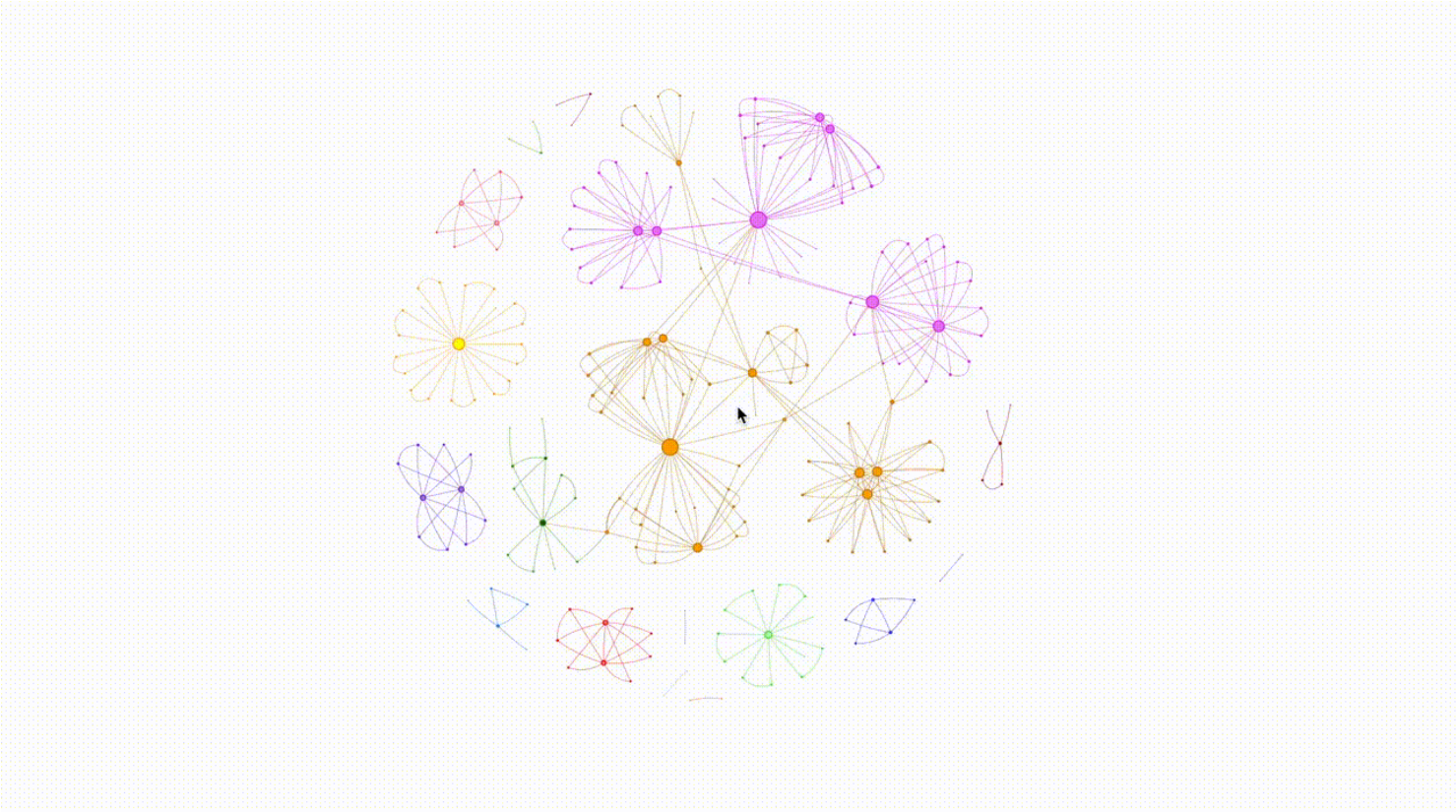
All it Takes is a Few Lines of Code

towardsdatascience.com

Pyvis has a built-in NetworkX Helper to translate our NetworkX graph into PyVis Objects. So we need no more coding.... Yay!!

Remember, we have already calculated the weights of each edge for the thickness of the edge, the communities of nodes for their colour, and the degree of each node as their size.

So, with all these bells and whistles, here is our graph.



Gif generate by the author using the project discussed in this article.

Link to the interactive graph: *https://rahulnyk.github.io/knowledge_graph/*

We can zoom in and out and move nodes and edges as we wish. We also have slider pannel at the bottom of the page to change the physics of the graph. See how the graph can help us ask the right questions and understand the subject matter better!

We can further discuss how our graph can help us build Graph Augmented Retrieval and how that can help us build a better RAG pipeline. But I think it's better to leave it for another day. We have achieved our goal for this article already!

Github Repo

GitHub - rahulnyk/knowledge_graph: Convert any text to a graph of knowledge. This can be used for...

Convert any text to a graph of knowledge. This can be used for Graph Augmented Generation or Knowledge Graph based QnA...

github.com

Contributions and suggestions most welcome

I have used the following article for the demonstration of my code.

Saxena S G, Godfrey T (June 11, 2023) India's Opportunity to Address Human Resource Challenges in Healthcare. Cureus 15(6): e40274. DOI 10.7759/cureus.40274

I am grateful to the authors for the wonderful work, and for releasing it under the Creative Commons Attribution License.

About me

I am a learner of architecture (not the buildings… the tech kind). In the past, I have worked with Semiconductor modelling, Digital circuit design, Electronic Interface modelling, and the Internet of Things. Currently, I am working with Data Interoperability and data warehouse architectures for Health and Wellness, at Walmart Health and Wellness.

Knowledge Graph | NLP | Large Language Models | Open Source | Hands On Tutorials