Exploring GraphRAG: From Local to Global with Implementation

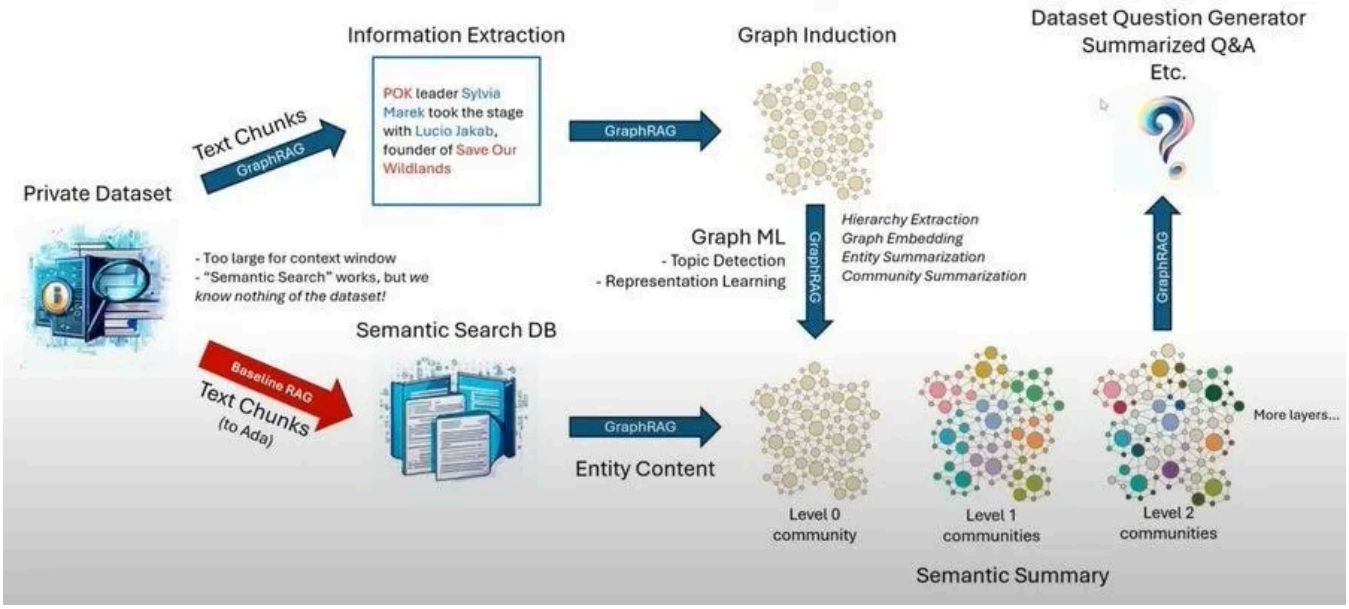
Rishi · Follow

11 min read · Aug 1, 2024

Introduction

We've explored RAGs in the previous blogs, and today we'll be focusing on a new approach — GraphRAG. After the introduction of LLMs, RAG quickly became a significant development in the generative AI space. However, these systems also introduced new challenges, each requiring unique solutions. Over the past year, numerous strategies have been developed to make RAG systems more stable and dynamic. In a similar effort, Microsoft Research recently announced GraphRAG, which provides a structured, graph-based representation, enhancing the retrieval and generation process. This innovative approach not only improves the accuracy and relevance of responses but also offers a more intuitive way to visualize and interact with the underlying data.

With this exciting new tool in our hands, let's dive in and explore how GraphRAG can be used for your data processing workflows!



What is GraphRAG?

GraphRAG, short for Graph Retrieval-Augmented Generation, is an advanced approach combining graph neural networks and retrieval-augmented generation techniques. It's a knowledge-enabled RAG approach that retrieves information from a knowledge graph for a given query. This method enhances the ability of language models by leveraging structured knowledge from graph databases and external information sources.

Why GraphRAG?

Traditional language models have made significant strides in understanding and generating human language. However, they often struggle with incorporating structured knowledge and retrieving relevant information from vast datasets. RAG struggles with global questions directed at an entire text corpus, such as "What are the main themes in the dataset?" This is a query-focused summarization (QFS) task rather than an explicit retrieval task. GraphRAG aims to solve exactly this kind of task. GraphRAG addresses these challenges by integrating GNNs, which excel at handling structured data, with retrieval-augmented generation techniques that fetch and incorporate external information in real-time.

How It Works

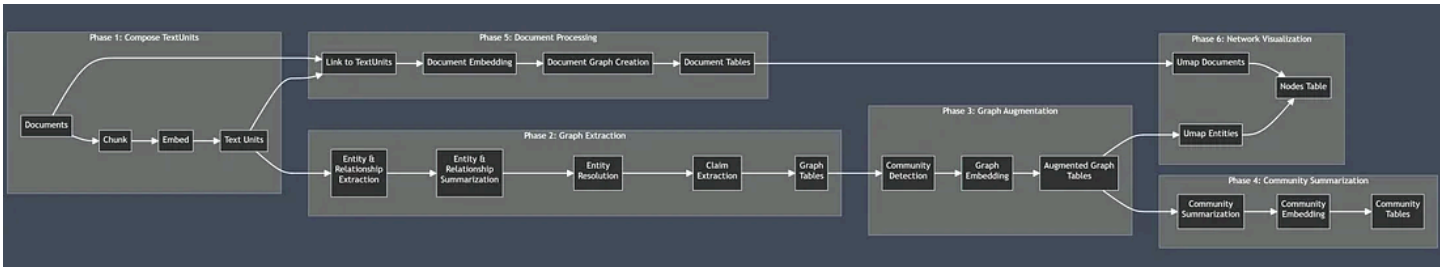
At a higher level, GraphRAG starts with source documents containing various information. These documents are processed using a Large Language Model (LLM) to extract structured information about entities and their relationships. This extracted data is then used to build a knowledge graph.

Once the knowledge graph is ready, GraphRAG uses a mix of graph algorithms, like the Leiden community detection algorithm, and LLM prompting to generate natural language summaries. These summaries provide insights into the communities of entities and relationships found within the knowledge graph.

Process

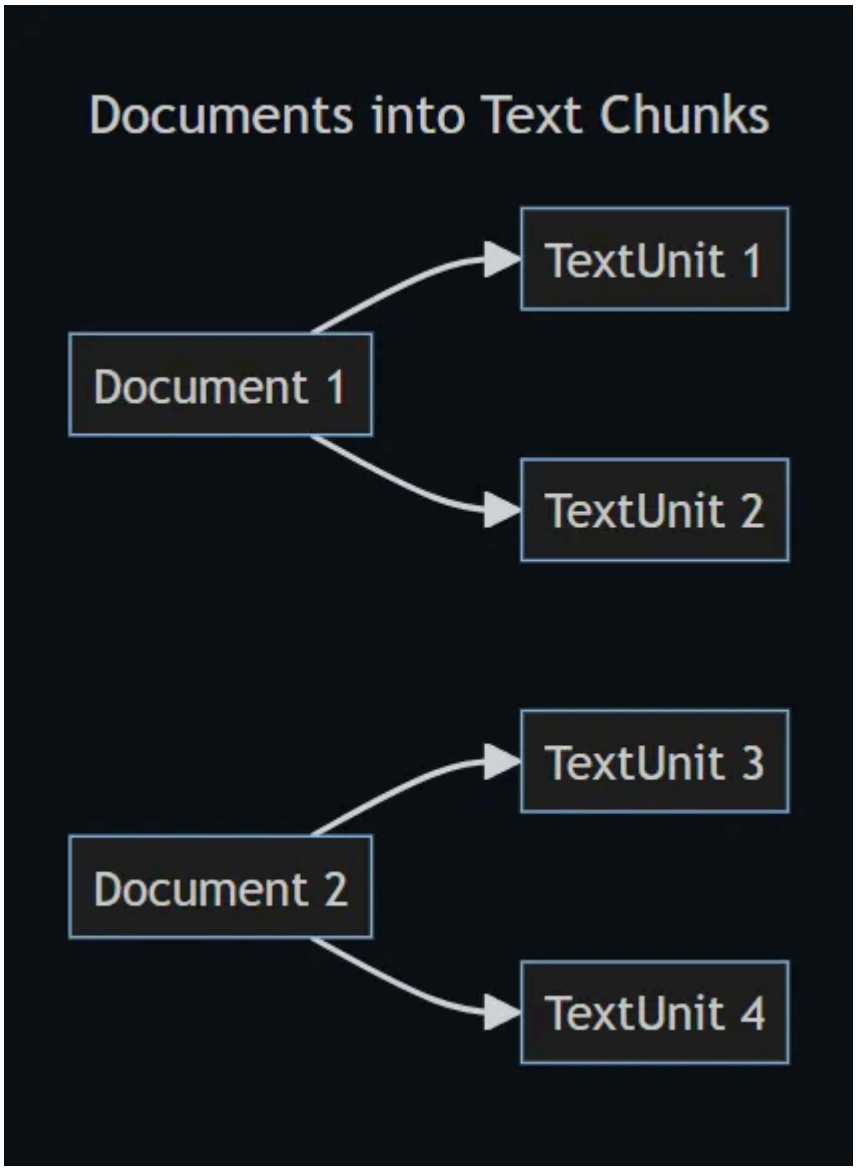
GraphRAG utilizes advanced machine learning to enhance data processing and retrieval. There are 2 main structural aspect — Indexing and Querying. Here's a simplified overview of how it works:

Indexing



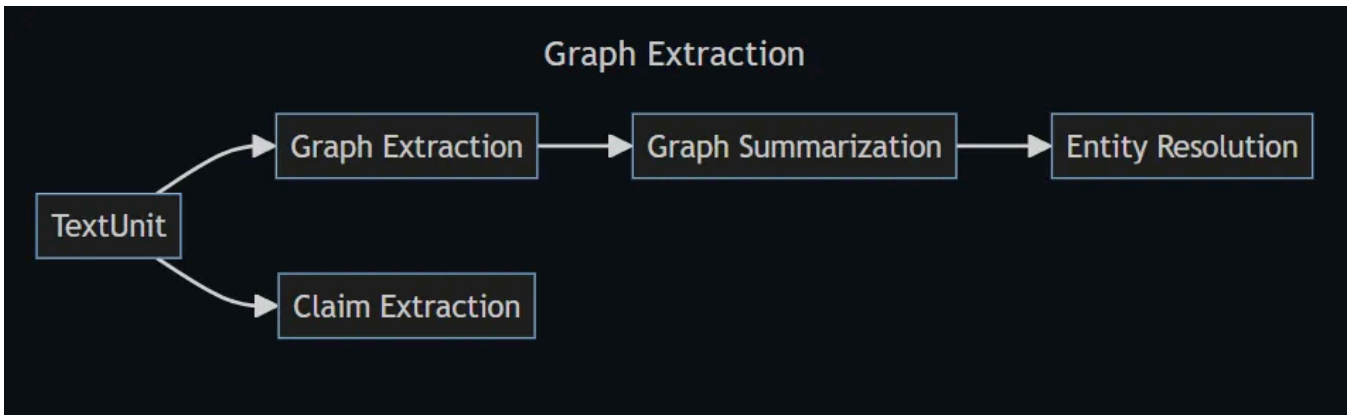
> *Creating Text Units:*

- We break down the input text into smaller chunks called TextUnits.



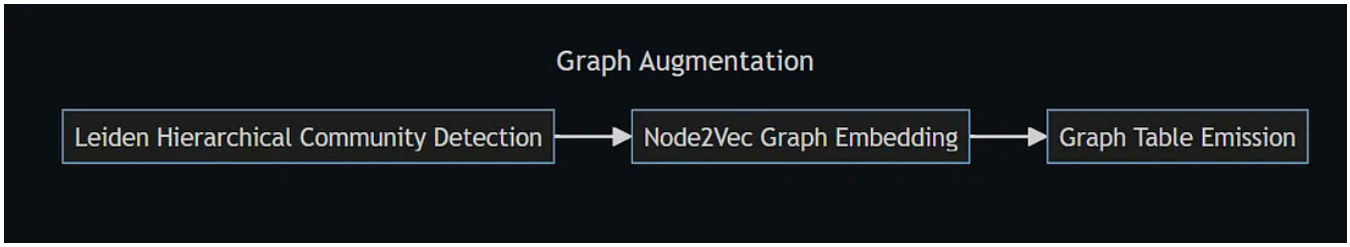
> *Extracting Information:*

- A Large Language Model (LLM) scans these TextUnits to identify and extract entities, relationships, and key claims.



> *Building the Graph:*

- With the extracted information, a graph is created.
- This graph is then organized into clusters using the Leiden algorithm.
- In the graph, each node represents an entity, with its size showing how connected it is and its color indicating which community it belongs to.



> *Summarizing Communities:*

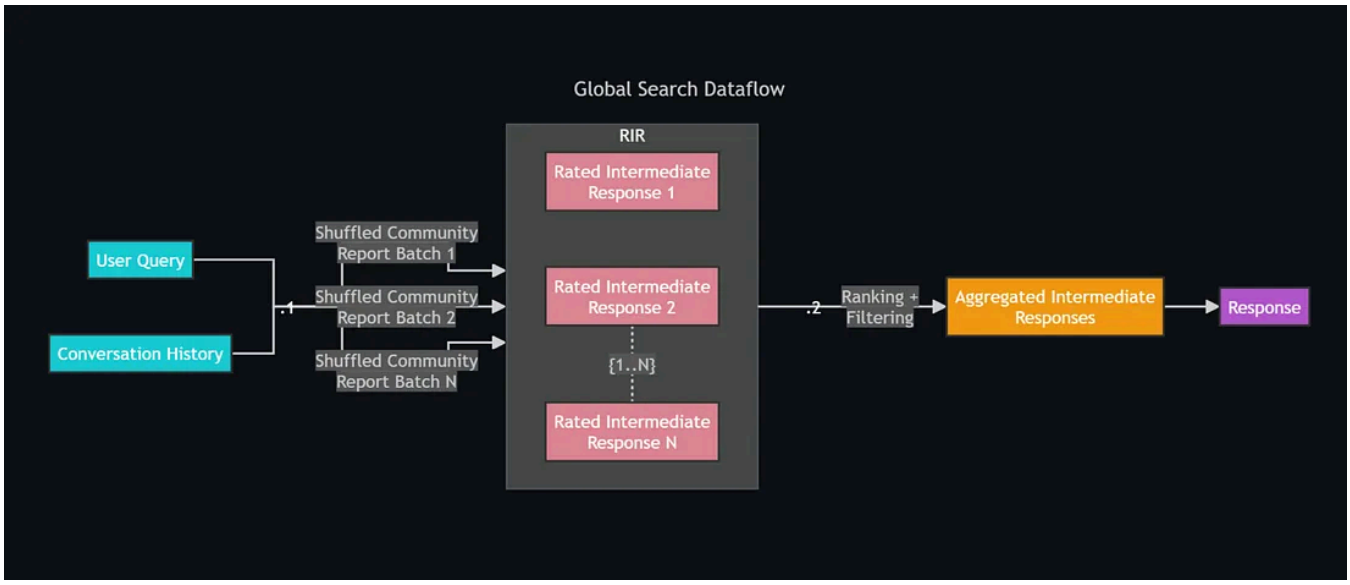
- Summaries are generated for each community within the graph, starting from the smallest units and building up.
- This helps to get a comprehensive understanding of the dataset.

Querying

When you need to query the system, the structured information from the graph is used to provide relevant context for answering questions. There are two main ways to query:

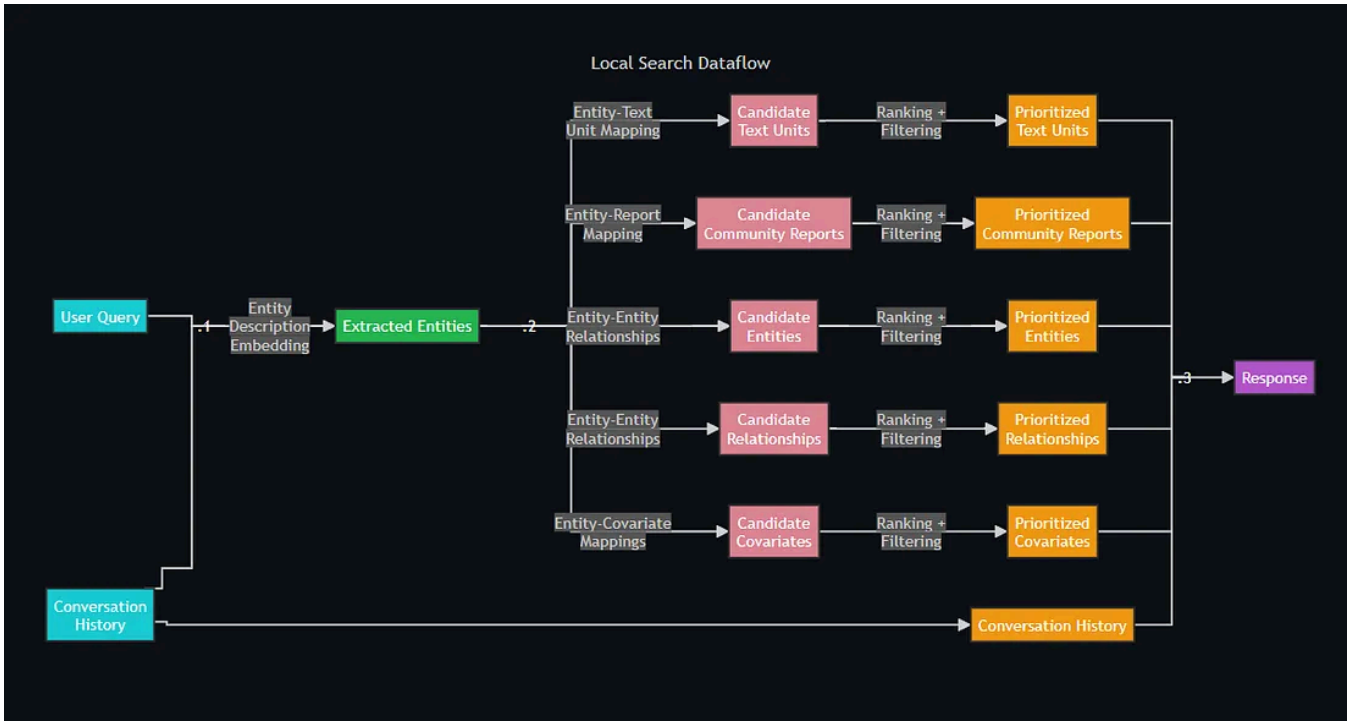
> *Global Search:*

- For broad questions about the entire dataset, leveraging the community summaries to get a big-picture answer.



> *Local Search:*

- For specific questions about individual entities, looking at their connections and related concepts.



Here is the detailed guide if you want dive deeper into the process: https://microsoft.github.io/graphrag/posts/index/1-default_dataflow/

We have dived into the idea behind GraphRAG, and it's time to move on to the practical aspect — implementation. Let's leverage our knowledge and explore how to put it into practice, step by step. Ready to dive in? Let's go!

Implementation

Here, we will utilize the repository I have created for deploying the model locally in order to make use of OpenAI-compatible endpoints: Open LLM Server

Before diving in, I want to let you know that I've created a Colab notebook for easy access. You can find it here: https://colab.research.google.com/drive/1uhFDnih1WKrSRQHisU-L6xw6coapgR51?usp=sharing

Setup Endpoint

To begin, we need to set up the Open LLM Server by following these steps:

Step 1: Clone the Repository

- To kick things off, we need to grab the Open LLM Server repository from GitHub.
- This repository contains all the necessary code and resources to get our server up and running.

```
git clone https://github.com/rushizirpe/open-llm-server.git
```

Step 2: Install Dependencies

- After cloning the Open LLM Server repository, the next step is to navigate into the directory and install the required dependencies.
- Without these dependencies, we won't be able to run the server or execute any API requests.

```
cd open-llm-server
pip install -q -r requirements.txt
```

Step 3: Launch the Server

- Once the dependencies are installed, it's time to launch the server.
- Running the following command will start the server, allowing it to listen for incoming API requests.

```
python scripts/launch.py start --host 127.0.0.1 --port 8888 --reload
```

You can also use docker to launch the server. Here's how: https://github.com/rushizirpe/open-llm-server?tab=readme-ov-file#dockerhub

Step 4: Verify Connection

- Once the server is running, let's make sure everything is working as it should.
- This is required, as the server must be operational for you to interact with it and utilize the GraphRAG functionalities.
- We can do this by sending a quick test request to the server's API. If the server responds, we're all set to go!

```
curl http://127.0.0.1:8888/
```


```
{
    "status": "System Status: Operational",
    "gpu": "Available",
    "gpu_details": {
        "GPU 0": {
            "compute_capability": "(8, 9)",
            "device_name": "NVIDIA L4"
            }
    }
}
```


> *Troubleshooting*

If you run into any hiccups along the way, here are some common errors and how to fix them:

- `curl: (7) Failed to connect to to localhost port 8888 after 0 ms: Connection Refused`

  This usually means that the server isn't up and running. No worries — just double-check that you've launched it successfully!

- `curl: (52) Empty reply from server`

  This indicates that there's not enough RAM to load the LLM model specified in the configuration file. If you see this message, try allocating more memory or adjusting your model settings.

And that's it! You should now have a running server and can interact with it using API requests as needed for GraphRAG.

GraphRAG

Now that we have our Server running, it's time to set up GraphRAG so we can index and query questions.

Step 1: Install

- First, we need to install the GraphRAG package.
- This is where we bring in the tools we'll need for our indexing and querying tasks.
- It's a quick and easy step — just run the following command:

```
pip install -q graphrag
```

Step 2: Initialization

- With GraphRAG installed, let's create the necessary directory structure and initialize our project.
- First, we will set up the environment where all our data will be stored and processed. And then we will start indexing.
- Don't worry if you see some warnings related to CUDA or TensorFlow; as long as the initialization finishes successfully, you're good to go!

```
cd ..
mkdir -p ./ragtest/input
python -m graphrag.index - init - root ./ragtest
```

Step 3: Configuration (YAML)

- Next, we'll update the YAML configuration file that was generated automatically during the initialization of GraphRAG.
- This file serves as a blueprint for how our indexing and querying will function, containing settings that instruct GraphRAG on handling our data.
- Make sure to replace `api_base` to `http://localhost:8888/v1` if you want to use a local endpoint. Otherwise, use `GROQ_API_KEY` as mentioned in Step 5.

```
#./ragtest/settings.yaml

encoding_model: cl100k_base
skip_workflows: []
llm:
  api_key: ${GROQ_API_KEY}
  type: openai_chat # or azure_openai_chat
  model:  llama3-8b-8192
  model_supports_json: true # recommended if this is available for your model.
  max_tokens: 3000
  # request_timeout: 180.0
  api_base: https://api.groq.com/openai/v1
  # api_version: 2024-02-15-preview
  # organization: <organization_id>
  # deployment_name: <azure_model_deployment_name>
  tokens_per_minute: 6000 # set a leaky bucket throttle
  requests_per_minute: 2 # set a leaky bucket throttle
  max_retries: 3
  # max_retry_wait: 10.0
  # sleep_on_rate_limit_recommendation: true # whether to sleep when azure sugg
  # concurrent_requests: 25 # the number of parallel inflight requests that may

parallelization:
  stagger: 0.3
  # num_threads: 50 # the number of threads to use for parallel processing

async_mode: threaded # or asyncio

embeddings:
  ## parallelization: override the global parallelization settings for embeddir
  async_mode: threaded # or asyncio
```

```
  llm:
    api_key: ${GRAPHRAG_API_KEY}
    # type: openai_embedding # or azure_openai_embedding
    model: nomic-ai/nomic-embed-text-v1.5
    api_base: http://127.0.0.1:1234/v1
    # api_version: 2024-02-15-preview
    # organization: <organization_id>
    # deployment_name: <azure_model_deployment_name>
    # tokens_per_minute: 150_000 # set a leaky bucket throttle
    # requests_per_minute: 10_000 # set a leaky bucket throttle
    # max_retries: 10
    # max_retry_wait: 10.0
    # sleep_on_rate_limit_recommendation: true # whether to sleep when azure su
    # concurrent_requests: 25 # the number of parallel inflight requests that m
    # batch_size: 16 # the number of documents to send in a single request
    # batch_max_tokens: 8191 # the maximum number of tokens to send in a single
    # target: required # or optional



chunks:
  size: 300
  overlap: 100
  group_by_columns: [id] # by default, we don't allow chunks to cross documents

input:
  type: file # or blob
  file_type: text # or csv
  base_dir: "input"
  file_encoding: utf-8
  file_pattern: ".*\\.txt$"

cache:
  type: file # or blob
  base_dir: "cache"
  # connection_string: <azure_blob_storage_connection_string>
  # container_name: <azure_blob_storage_container_name>

storage:
  type: file # or blob
  base_dir: "output/${timestamp}/artifacts"
  # connection_string: <azure_blob_storage_connection_string>
  # container_name: <azure_blob_storage_container_name>

reporting:
  type: file # or console, blob
  base_dir: "output/${timestamp}/reports"
  # connection_string: <azure_blob_storage_connection_string>
  # container_name: <azure_blob_storage_container_name>

entity_extraction:
  ## llm: override the global llm settings for this task
  ## parallelization: override the global parallelization settings for this tas
```

```
    ## async_mode: override the global async_mode settings for this task
    prompt: "prompts/entity_extraction.txt"
    entity_types: [organization,person,geo,event]
    max_gleanings: 0

  summarize_descriptions:
    ## llm: override the global llm settings for this task
    ## parallelization: override the global parallelization settings for this tas
    ## async_mode: override the global async_mode settings for this task
    prompt: "prompts/summarize_descriptions.txt"
    max_length: 500

  claim_extraction:
    ## llm: override the global llm settings for this task
    ## parallelization: override the global parallelization settings for this tas
    ## async_mode: override the global async_mode settings for this task
    # enabled: true
    prompt: "prompts/claim_extraction.txt"
    description: "Any claims or facts that could be relevant to information disco
    max_gleanings: 0

  community_report:
    ## llm: override the global llm settings for this task
    ## parallelization: override the global parallelization settings for this tas
    ## async_mode: override the global async_mode settings for this task
    prompt: "prompts/community_report.txt"
    max_length: 2000
    max_input_length: 8000

  cluster_graph:
    max_cluster_size: 10

  embed_graph:
    enabled: false # if true, will generate node2vec embeddings for nodes
    # num_walks: 10
    # walk_length: 40
    # window_size: 2
    # iterations: 3
    # random_seed: 597832

  umap:
    enabled: false # if true, will generate UMAP embeddings for nodes

  snapshots:
    graphml: false
    raw_entities: false
    top_level_nodes: false

  local_search:
    # text_unit_prop: 0.5
    # community_prop: 0.1
    # conversation_history_max_turns: 5
    # top_k_mapped_entities: 10
```

```
    # top_k_relationships: 10
    # max_tokens: 12000

  global_search:
    # max_tokens: 12000
    # data_max_tokens: 12000
    # map_max_tokens: 1000
    # reduce_max_tokens: 2000
    # concurrency: 32
```

Step 4: Sample Input

- Now, let's create a sample input text file in `./ragtest/input/sample.txt`.
- This is where we'll put the data that we want to index and query.
- In this example, we're using a snippet from "Marley's Ghost" from Project Gutenberg.

```
MARLEY'S GHOST
Marley was dead, to begin with. There is no doubt whatever about that.
The register of his burial was signed by the clergyman, the clerk, the
undertaker, and the chief mourner. Scrooge signed it. And Scrooge's name was gc
```

- Feel free to modify this input data to try with different!

Step 5: Set API Keys

- Before we dive into indexing, we need to set our API keys.
- These keys are optional but can help speed up access to the models and resources we'll be working with.

Here's what you need to know:

1. **GROQ API Key (Optional)**

- If you have a high-end GPU, consider using local endpoints instead of GROQ. This allows you to handle longer text inputs.
- You can get your free `GROQ_API_KEY` from here.

2. Hugging Face Token (Optional)

- If you're using models from a gated repository on Hugging Face, you'll need a Hugging Face token.
- You can obtain your token from here.

> *Using Local Endpoints*

- If you opt for local endpoints, you don't need the GROQ API key.

Ensure you set up your API keys correctly to use these endpoints.

```
export $GROQ_API_KEY = '<GROQ_API_KEY>'
export $HUGGING_FACE_TOKEN = '<HUGGING_FACE_TOKEN>'
```

Indexing

- Now we can run the indexing process, which analyzes the input text and prepares it for querying.
- This step will output the progress of various workflows, and you'll see a message indicating when everything has been completed successfully.

```
python -m graphrag.index - root ./ragtest
```

Query

- Now that we've successfully indexed our data, it's time to put it to the test by performing some queries!
- Let's explore two types of searches: Global Search and Local Search.

> *Global Search:*

- Global Search allows us to ask broader questions about the text.
- It looks through the entire dataset to find relevant information.
- Here's how you can perform a global search:

```
python -m graphrag.query - root ./ragtest - method global "Who is Marley?"
```

- *Output*:

```
SUCCESS: Global Search Response: **Marley's Significance**

Marley, a deceased individual, has a profound impact on Scrooge, who is deeply

**Marley's Burial and Legacy**

Marley's burial and legacy play a crucial role in shaping Scrooge's emotional s

**Key Takeaways**

In summary, Marley is a deceased individual who has a significant emotional imp

Note: The reports provided do not contain additional information about Marley b
```

> *Local Search*

- Local Search is great for digging into specific entities and their details.
- It focuses on particular sections of the text to give more detailed answers.
- Here's how you can perform a local search:

```
python -m graphrag.query - root ./ragtest - method local "Who signed the burial
```

You can ask more questions to get a sense of it.

With these steps, you should have a solid setup for using GraphRAG to index and query from data.

Wrapping Up

We've come a long way in this tutorial, from understanding the theory behind GraphRAG to implementing it step-by-step.

I hope this guide has made the process clear and straightforward. Feel free to experiment with different configuration and data to fully explore the potential of GraphRAG.

Thank you for following along, and happy learning! If you have any questions or run into issues, don't hesitate to reach out. Till Next Time!

> "Alone we can do so little, together we can do so much!"

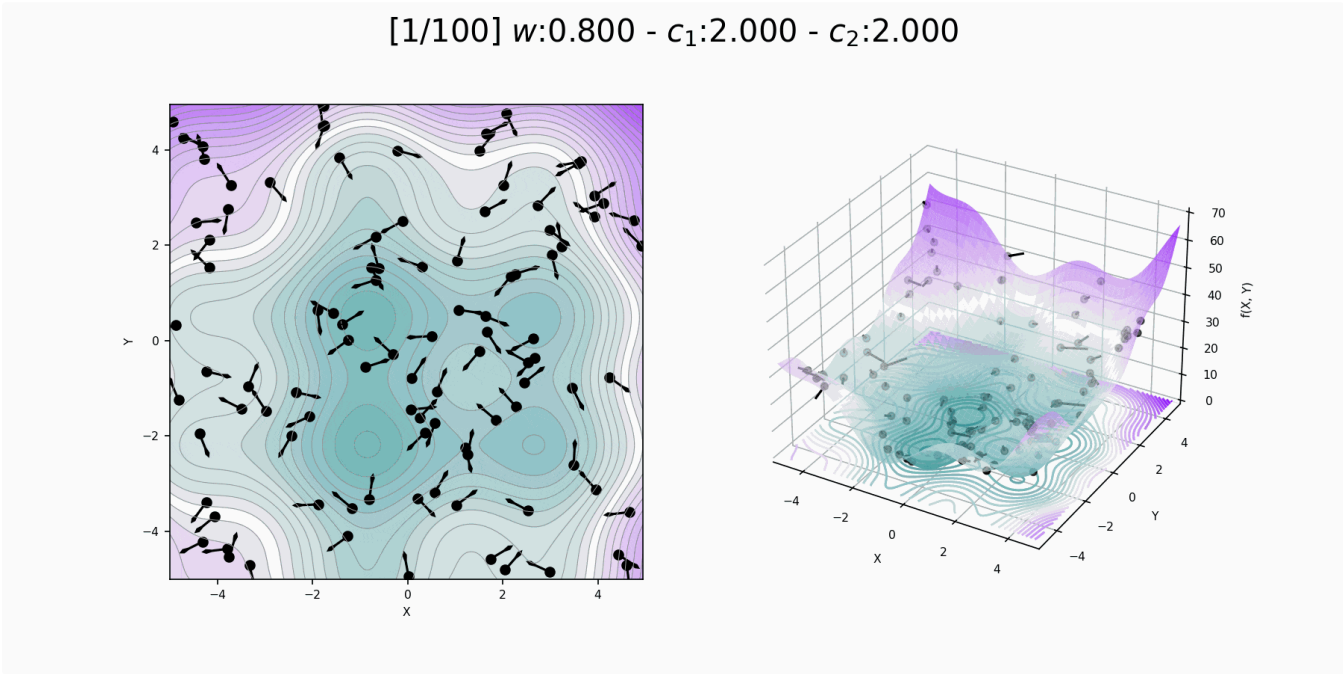
Large Language Models | NLP | Machine Learning | AI

Follow

Written by Rishi

31 Followers

More from Rishi

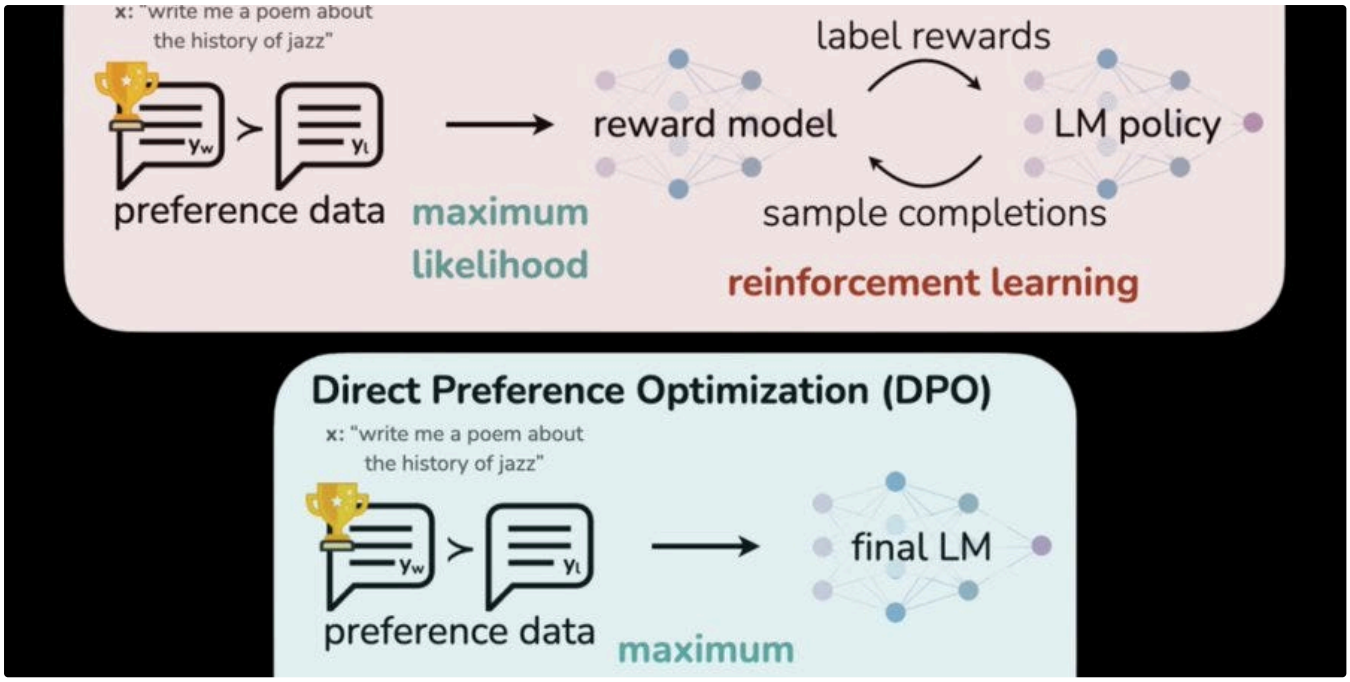


Rishi

Understanding Particle Swarm Optimization (PSO): From Basics to Brilliance

Optimizing complex functions can be a daunting task, but there's an algorithm that can make the process easier - Particle Swarm...

Mar 20 55

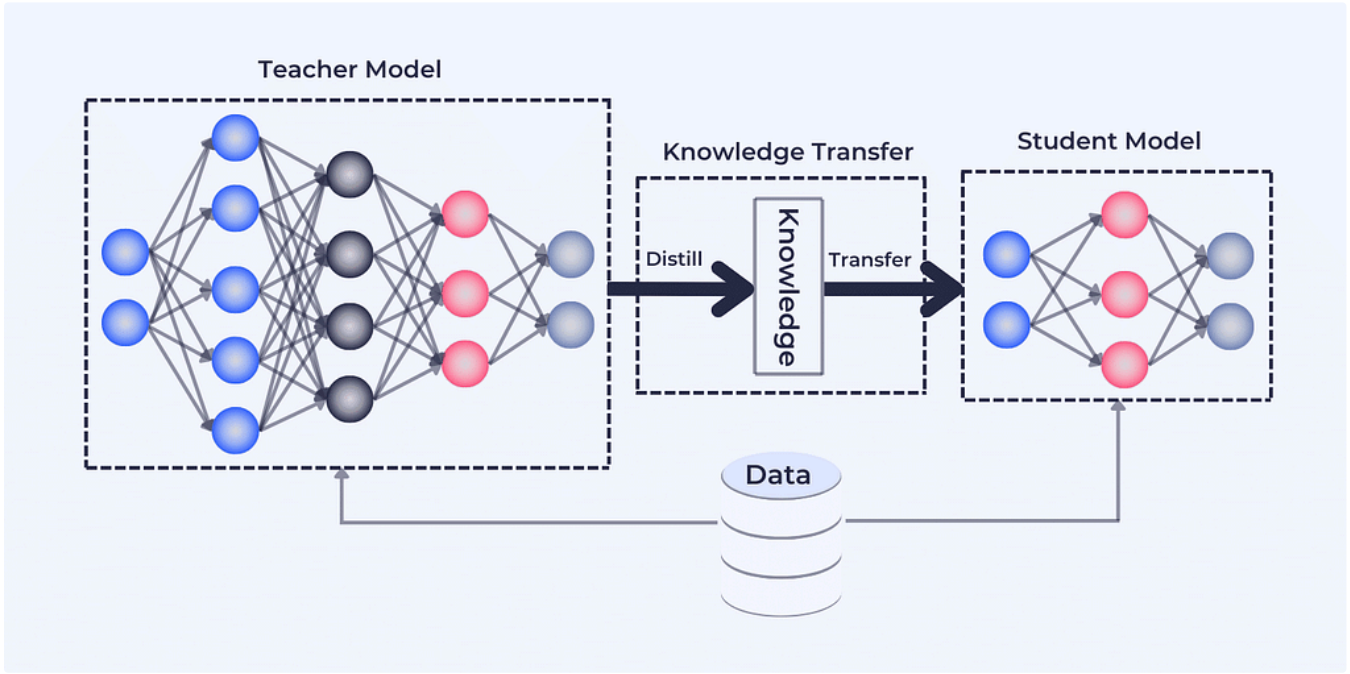


Rishi

Direct Preference Optimization (DPO) in LLMs

Guide to Direct Preference Optimization with implementation

Jan 18 4

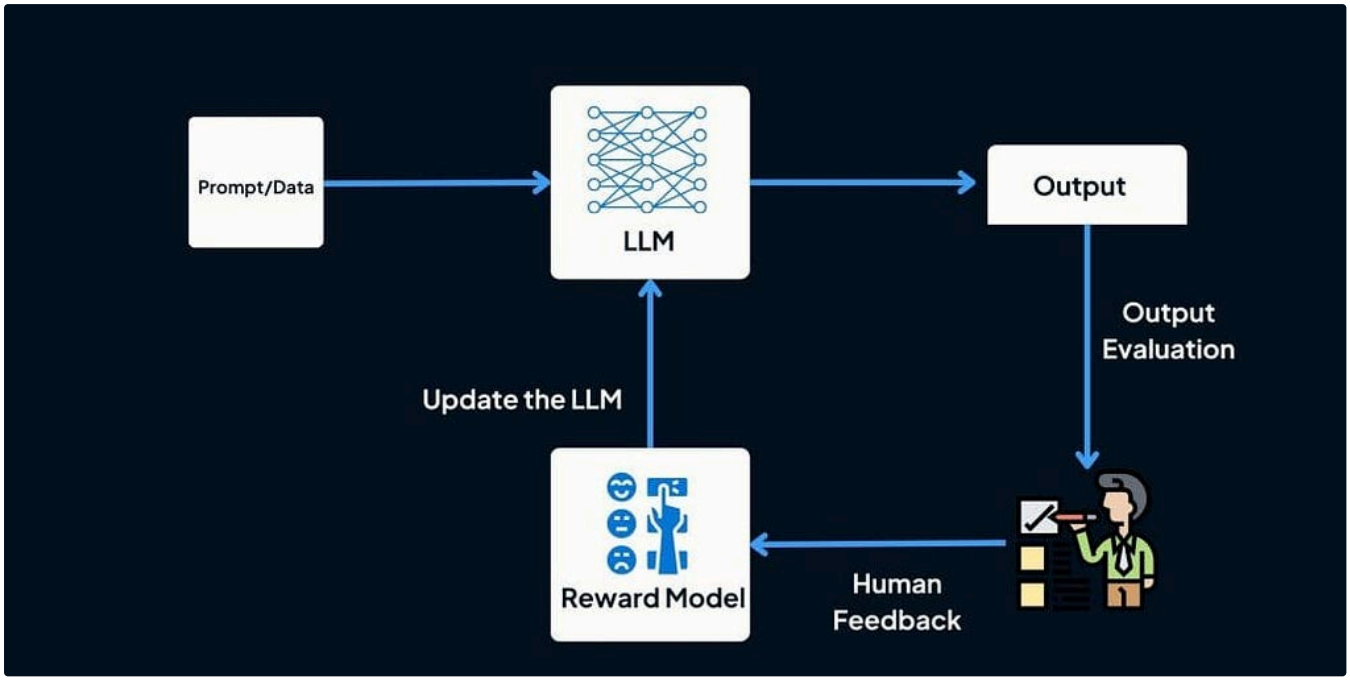


Rishi

The Power of Model Compression: Guide to Pruning, Quantization, and Distillation in Machine...

In today's world of AI, we need models that work fast and don't use up too many resources. To make this happen, we use model compression...

Jan 30 10



Rishi

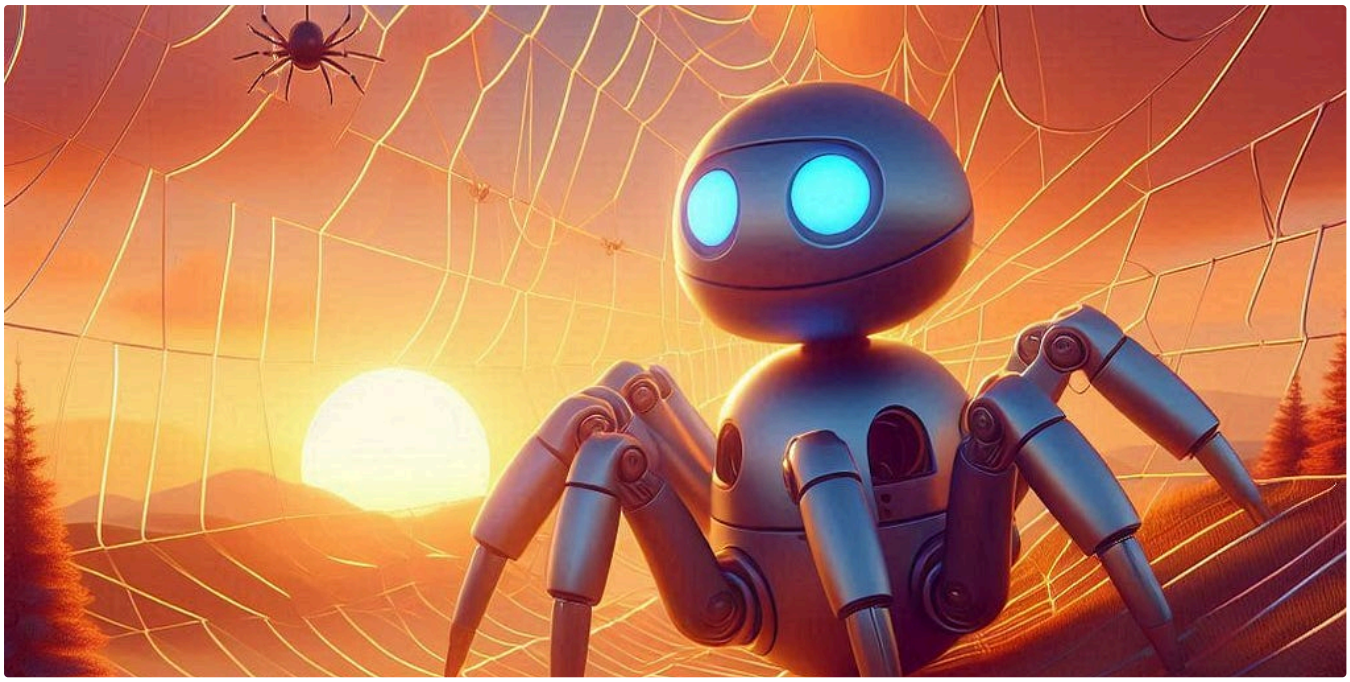
Reinforcement Learning with Human Feedback in LLMs: A Comprehensive Guide

Guide to RLHF in Large Language Models (LLMs)

Jan 11 16 1

See all from Rishi

Recommended from Medium

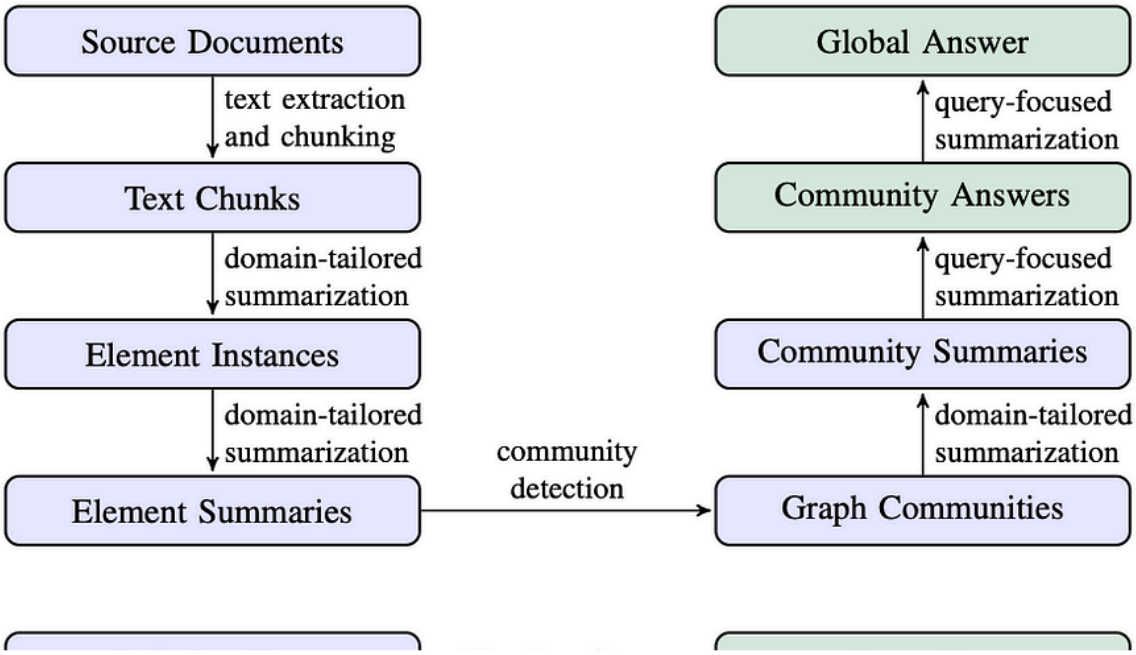


Valentina Alto in Microsoft Azure

Introducing GraphRAG with LangChain and Neo4j

Part 1: Getting Started with Graph Databases in the LLMs Era

Apr 28 836 14



Zilliz

GraphRAG Explained: Enhancing RAG with Knowledge Graphs

Introduction to RAG and Its Challenges

Aug 7 68 1

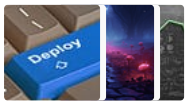
Lists

Natural Language Processing

1741 stories · 1324 saves

The New Chatbots: ChatGPT, Bard, and Beyond

12 stories · 471 saves



Predictive Modeling w/ Python

20 stories · 1579 saves

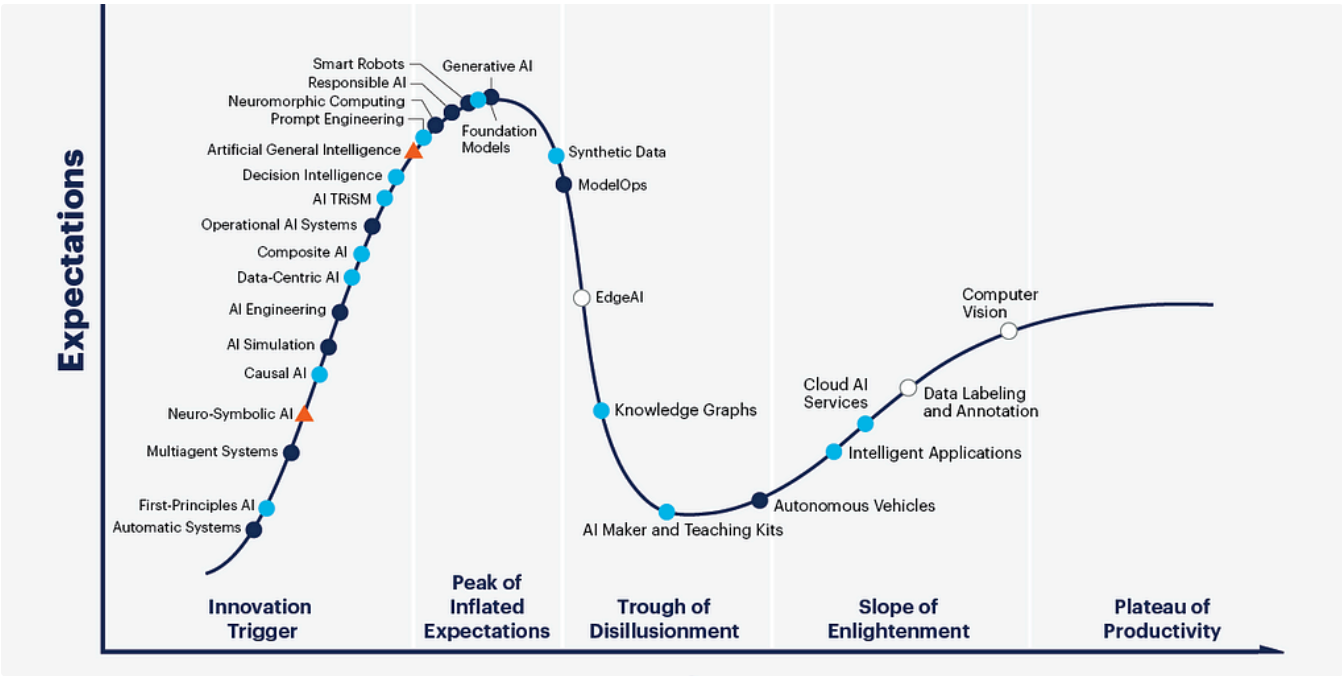
Practical Guides to Machine Learning

10 stories · 1916 saves

Mark OBrien

Build GraphRAG Using Streamlit, LangChain, Neo4j & GPT-4o

Jul 4 101

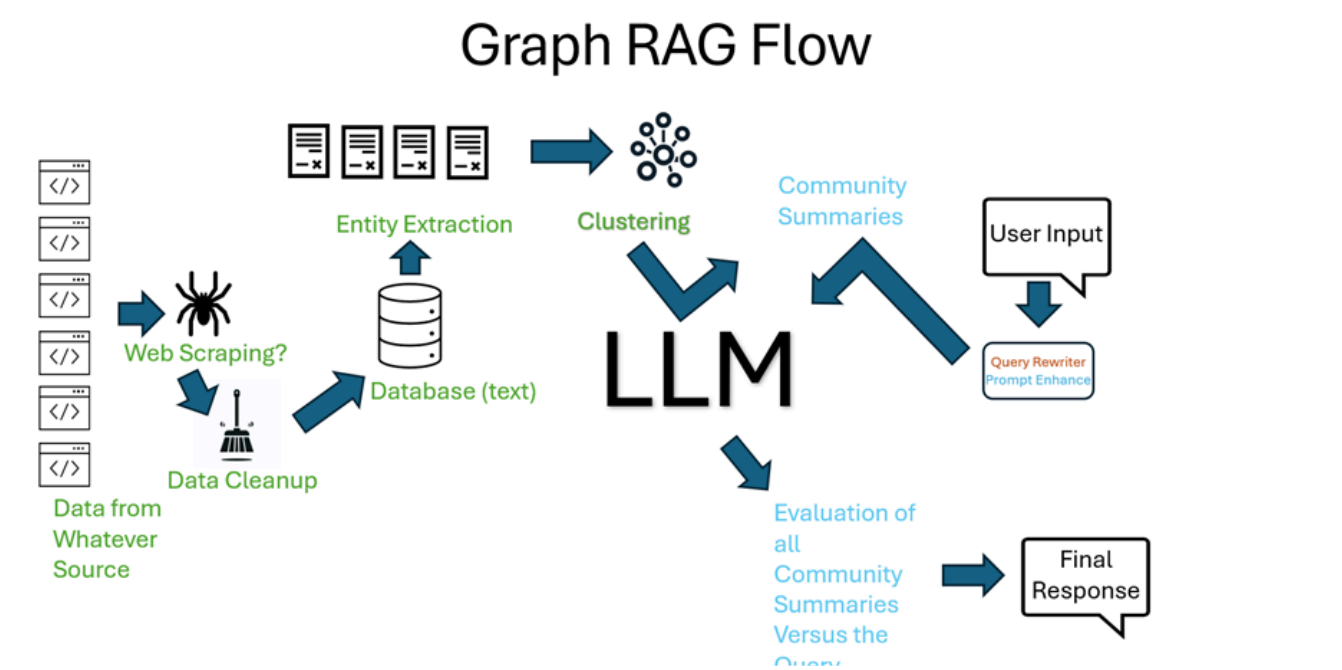


Vishal Rajput in AIGuys

Why GEN AI Boom Is Fading And What's Next?

Every technology has its hype and cool down period.

Sep 4 1.97K 60



Troyusrex in Generative AI

Graph RAG Has Awesome Potential, But Currently Has Serious Flaws

I was very excited when Llama Index came out with their new Graph RAG implementation. It has great potential to make my RAG AI platform...

Jul 19 192 1

Use Case Families	Generative Models	Non-Generative ML	Optimisation	Simulation	Rules	Graphs
Forecasting	Low	High	Low	High	Medium	Low
Planning	Low	Low	High	Medium	Medium	High
Decision Intelligence	Low	Medium	High	High	High	Medium
Autonomous System	Low	Medium	High	Medium	Medium	Low
Segmentation	Medium	High	Low	Low	High	High
Recommender	Medium	High	Medium	Low	Medium	High
Perception	Medium	High	Low	Low	Low	Low
Intelligent Automation	Medium	High	Low	Low	High	Medium
Anomaly Detection	Medium	High	Low	Medium	Medium	High
Content Generation	High	Low	Low	High	Low	Low
Chatbots	High	High	Low	Low	Medium	High

Christopher Tao in Towards AI

Do Not Use LLM or Generative AI For These Use Cases

Choose correct AI techniques for the right use case families

Aug 10 3.6K 39

See more recommendations