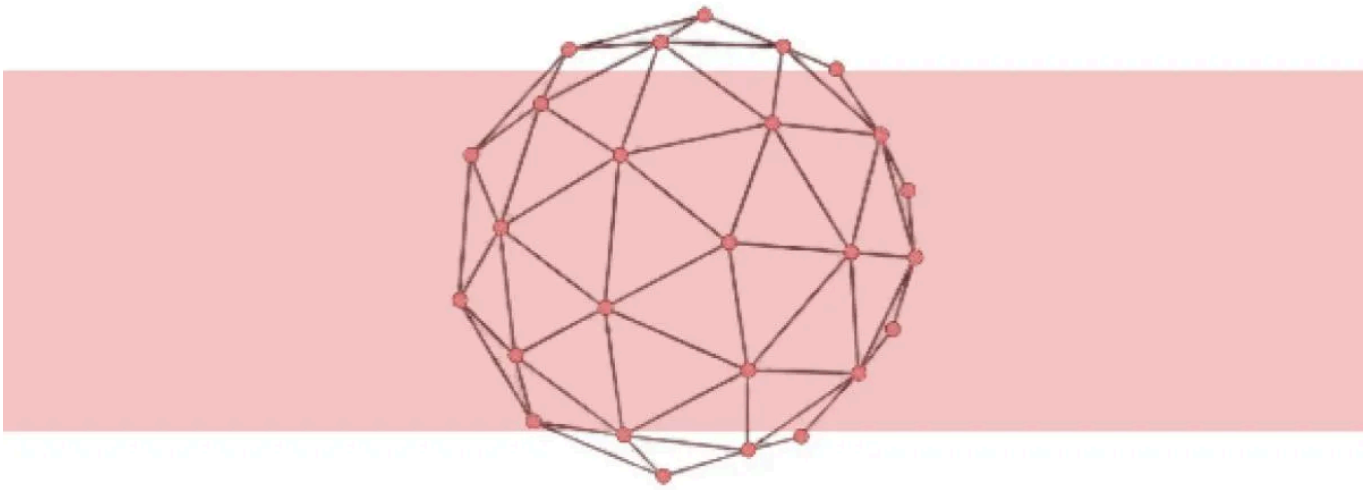


Illustration by the author

✦ Member-only story

From Text to Networks: The Revolutionary Impact of LLMs on Knowledge Graphs

A Step-by-Step Guide to Building and Leveraging Knowledge Graphs with LLMs

Lina Faik · Follow

Published in Towards Data Science · 15 min read · Aug 29, 2024

The rise of Large Language Models (LLMs) has revolutionized the way we extract information from text and interact with it. However, despite their impressive capabilities, LLMs face several inherent challenges, particularly in areas such as reasoning, consistency, and information's contextual accuracy. These difficulties come from the probabilistic nature of LLMs, which can lead to hallucinations, lack of transparency, and challenges in handling structured data.

This is where Knowledge Graphs (KGs) come into play. By integrating LLMs with KGs, AI-generated knowledge can be significantly enhanced. Why? KGs provide a structured and interconnected representation of information, reflecting the relationships and entities in the real world. Unlike traditional databases, KGs can capture and reason about the complexities of human knowledge, ensuring that the outputs of LLMs come from a structured, verifiable knowledge base. This integration leads to more accurate, consistent, and contextually relevant outcomes.

Industries like healthcare, finance, and legal services can greatly benefit from knowledge graphs due to their need for precise and interpretable information.

Objective

The article delves into the innovative ways LLMs can be harnessed to build and leverage knowledge graphs.

After reading this article, you will understand:

1. How to build a knowledge graph using LLMs

2. How LLMs can then leverage knowledge graphs to enhance their capabilities
3. What are the limitations and key considerations when integrating LLMs with knowledge graphs?

> No prior knowledge is required to understand the article. The experimentations described in the article were carried out focus only the libraries llama-index.

You can find the code here on GitHub.

1. Navigating Knowledge Graphs & LLMs Landscape

This section provides a comprehensive overview of the process involved in building and leveraging knowledge graphs with LLMs. It introduces the key concepts, terminology, and various approaches before delving into the specific methods in the following sections.

1.1. How does the process of developing and leveraging knowledge graphs unfold?

Figure 1 illustrates the overall workflow, consisting of two key steps:

1. Knowledge graph extraction, along with the possible techniques detailed in Section 2. This step results in the creation of the knowledge graph.
2. Retrieval using one of the techniques outlined, which will be discussed an in-depth in Section 3. These techniques can be employed to answer queries.

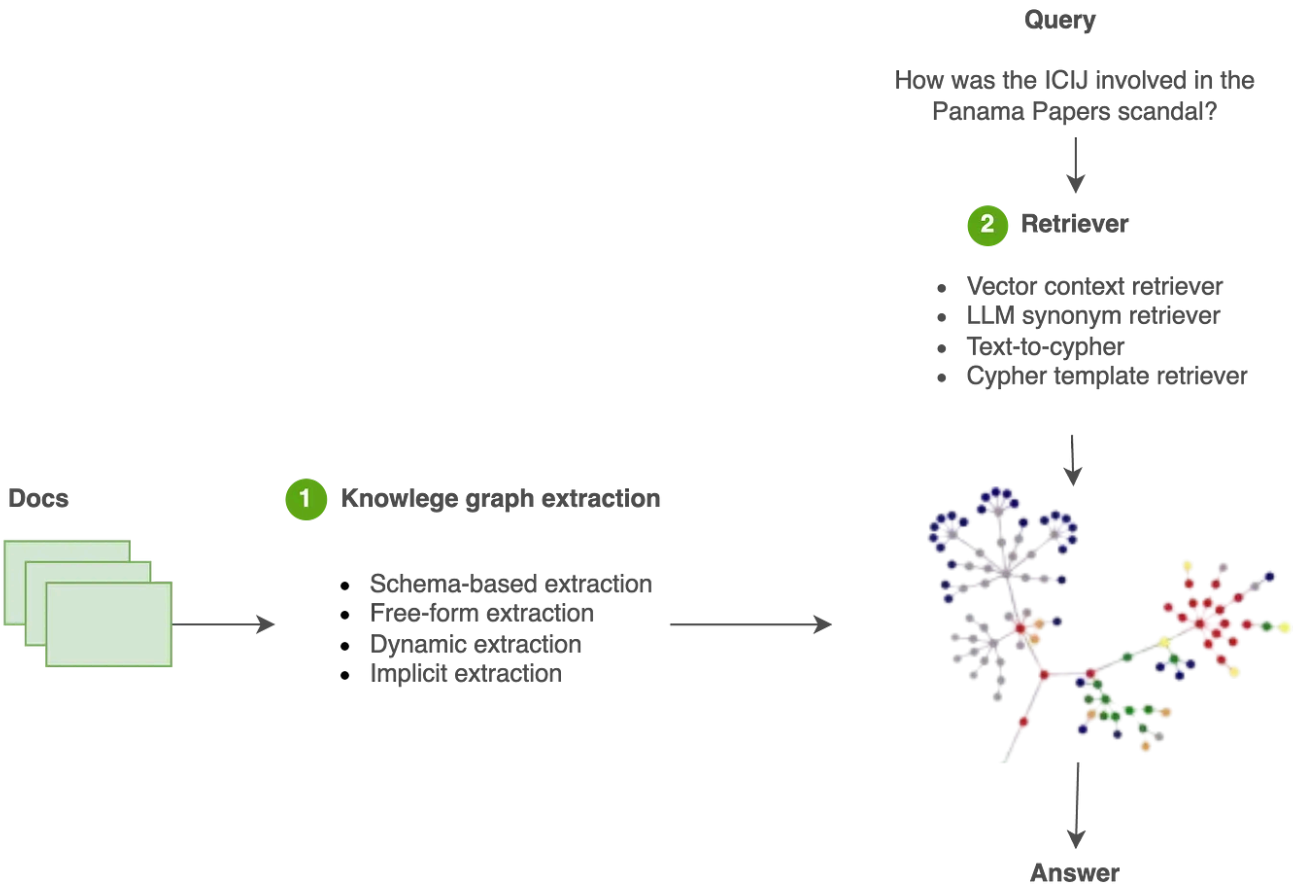


Fig 1 — Workflow

1.2. Refresher on knowledge graph terminology

In network terminology, a triplet refers to a fundamental unit of a knowledge graph. It consists of three elements: a subject (an entity), a relation (the connection between entities), and an object (another entity).

Nodes represent the entities (e.g., people, companies) within the network, while edges or links represent the relationships or interactions between these nodes.

Together, nodes and edges form a graph structure, where triplets define the specific connections within this network.

1.3. About Data

The data for the experiments is drawn from the Wikipedia page on the Panama Papers scandal.

2. Building Knowledge Graphs: Extraction Methods Explored

The library llama-index proposes 4 main methods to extract a knowledge graph from a corpus of documents using LLMs:

1. Schema-based extraction
2. Free-form extraction
3. Dynamic extraction
4. Implicit extraction

Let's delve into each of these methods.

2.1. Schema-based extraction

The main idea of this approach is to construct a graph based on a predefined network model. By specifying the desired entity types and their possible relationships, the LLM-based approach generates a network from the text, structured according to the specified model.

💡 **Example:** In the Panama Papers scandal, we can establish a predefined structure where entities include companies, banks, or scandals, and relationships are defined as 'involved' or 'owns.' The schema-based extracted graph would then be expected to reveal connections such as 'Person A owns Company B' or 'Bank D is involved in Scandal E.'

How to implement schema-based extraction in practice?

Here's a quick skeleton for implementing schema-based extraction in practice:

```
from typing import Literal
from llama_index.core.indices.property_graph import SchemaLLMPathExtractor
from llama_index.core import PropertyGraphIndex
from llama_index.embeddings.openai import OpenAIEmbedding
from llama_index.llms.openai import OpenAI

# Define the possible entity types for the knowledge graph
entities = Literal["PERSON", "COMPANY", "COUNTRY", "BANK", "SCANDAL"]

# Define the possible relations between the entities in the knowledge graph
relations = Literal["OWNS", "LOCATED_IN", "INVOLVED_IN"]

# Define the schema that outlines which entities can have which relations
schema = {
    "PERSON": ["OWNS", "LOCATED_IN", "INVOLVED_IN"],
    "COMPANY": ["OWNS", "LOCATED_IN", "INVOLVED_IN"],
    "COUNTRY": ["LOCATED_IN"],
    "BANK": ["LOCATED_IN", "INVOLVED_IN"],
    "SCANDAL": ["INVOLVED_IN"],
}

# Create an instance of SchemaLLMPathExtractor to extract paths based on the def
kg_extractor = SchemaLLMPathExtractor(
  llm=OpenAI(model=LLM_MODEL, temperature=TEMPERATURE), # Use OpenAI's language
  possible_entities=entities,  # Define the types of entities to extract
  possible_relations=relations,  # Define the types of relations to extract
  kg_validation_schema=schema,  # Use the predefined schema for validation
  strict=True,  # Enforce strict validation; only entities and relations defined
)

# Create a PropertyGraphIndex from the provided documents, using the specified e
index = PropertyGraphIndex.from_documents(
    documents,  # The input documents to be processed and indexed
    embed_model=OpenAIEmbedding(model_name=EMBEDDING_MODEL),  # Use OpenAI's emb
    show_progress=True,  # Display progress during indexing
    kg_extractors=[kg_extractor],  # Use the previously defined SchemaLLMPathExt
)

# Define the storage path for the keyword extractor
path_output_storage_kg_extractor = f"{path_output_storage}/{kw_extractor_name}/"

# Create the storage directory if it doesn't already exist
```

```
if not os.path.exists(path_output_storage_kg_extractor):
    os.makedirs(path_output_storage_kg_extractor)

# Persist the index's storage context to the specified directory
index.storage_context.persist(persist_dir=path_output_storage_kg_extractor)

# Save the knowledge graph as a NetworkX graph to an HTML file
index.property_graph_store.save_networkx_graph(name=f"{path_output}/kwnoledge_gr
```

> ⚠️ Note: In the `kg_extractors` argument of `PropertyGraphIndex`, multiple knowledge extractor instances can be added, enabling the combination of various extraction methods within a single network.

Figure 2 displays a part of the schema-based extracted graph generated by the code. The nodes represent either a person, company, country, bank, or scandal, while the relationships are limited to ownership and involvement, as predefined in the code.

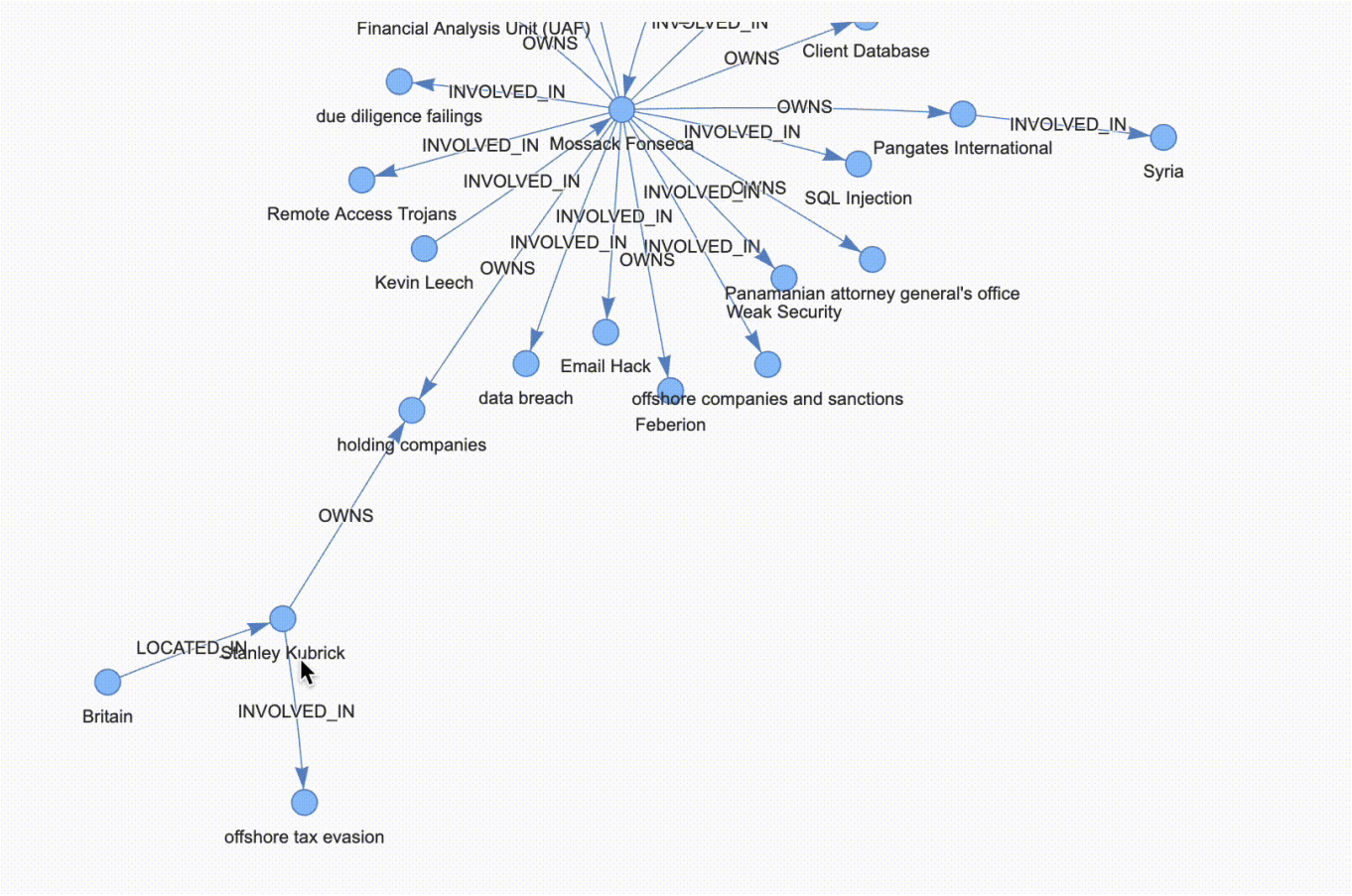


Fig 2 — Part of the schema-based extract graph

What is the method behind schema-based extraction?

✓ **Text analysis:** The extractor first analyzes the text content within each chunk using an LLM. This model interprets the text to identify potential entities (such as people, organizations, or locations) and relationships (such as ownership, location, or association) based on the provided schema.

✓ **Path extraction:**

- Using the schema as a guide, the tool extracts specific relationships between identified entities, forming triplets.
- A triplet is a set of three components: a subject entity, a relationship, and an object entity (e.g., “Person X owns Company Y”).

- The `max_triplets_per_chunk` parameter limits the number of triplets processed at a time, ensuring efficiency when handling large datasets.

✓ **Validation:** The extracted triplets are then validated against the schema to ensure they fit the predefined types of entities and relationships. If the schema is set to strict mode, only triplets that fully comply with the schema are retained.

✓ **Graph construction:** The validated triplets are then used to construct or expand a knowledge graph. Each triplet adds nodes (entities) and edges (relationships) to the graph, gradually building a network that reflects the information extracted from the text.

✓ **Metadata integration:** The tool can also incorporate metadata, which may include additional context or properties associated with entities and relationships (e.g., timestamps, source documents). This helps in enriching the graph with more detailed and relevant information.

2.2. Free-form extraction

Free-form extraction is a flexible method of extracting relationships, or "triples," from text data. Contrary to the scheme-based extraction methods, it doesn't not rely on a predefined schema or structure but leverages the capabilities of a language model (LLM) to identify meaningful connections between entities on its own.

💡 **Example:** In the context of the Panama Papers scandal, free-form extraction can analyze leaked documents to uncover hidden relationships between individuals, companies, and offshore entities. For instance, the model might extract triples such as "Company B is registered in Offshore Jurisdiction C", or "Law Firm D manages Company B". These connections,

inferred from the context rather than explicitly stated, help build a knowledge graph that maps the complex network of entities involved in the scandal.

How to implement free-form extraction in practice?

```
from llama_index.core.indices.property_graph import SimpleLLMPathExtractor
# Create an instance of SimpleLLMPathExtractor
kg_extractor = SimpleLLMPathExtractor(
    llm=OpenAI(model=LLM_MODEL, temperature=TEMPERATURE)
)
```

Figure 3 presents a portion of the free-form extracted graph. Here, the relationships between nodes are more flexible, allowing for greater exploration. However, this flexibility involves a trade-off between the desire to explore and the need for a concise, essential-focused representation.

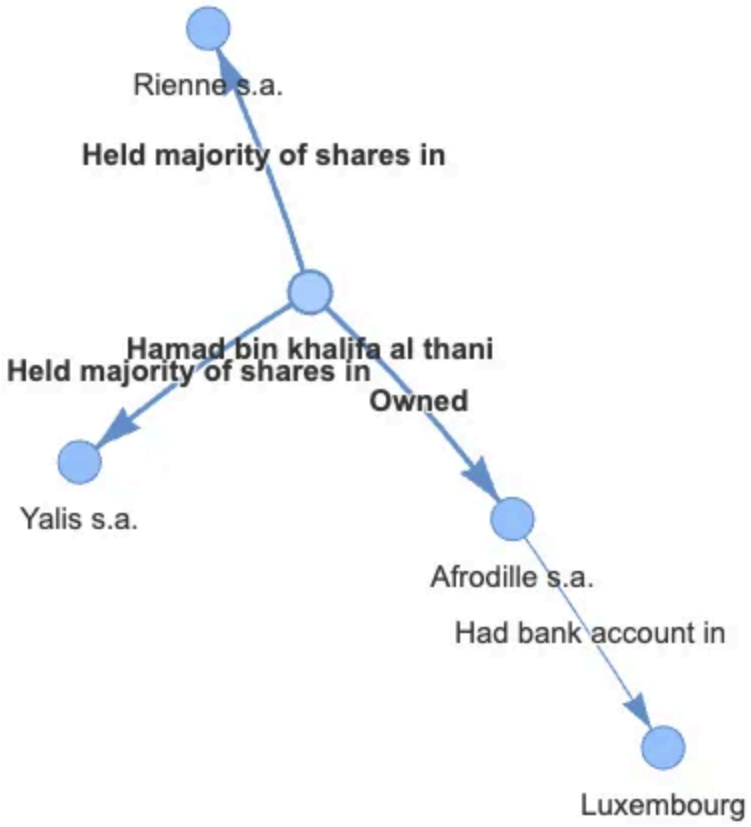


Fig 3 —Part of the free-from extracted graph

Source: “Former Emir of Qatar Hamad bin Khalifa Al Thani owned Afrodille S.A., which had a bank account in Luxembourg and shares in two South African companies. Al Thani also held a majority of the shares in Rienne S.A. and Yalis S.A., holding a term deposit with the Bank of China in Luxembourg. ” [3]

How does free-form extraction achieve its results?

✓ **Input text analysis:** The process begins with analyzing the input text from chunks. The content is passed through the language model, which uses a specific prompt to guide the extraction process. It generates potential triples. The extraction is controlled by the `max_paths_per_chunk` parameter,

which limits the number of triples generated in each chunk. This ensures the process remains efficient and manageable, especially when handling large volumes of text.

✓ **Parsing:** The output from the language model is then parsed by a function that extracts these triples from the raw text response.

✓ **Building the graph:** The extracted triples are then used to create or update a knowledge graph. These nodes and edges are enriched with metadata, which can include additional information like the source of the data or timestamps.

2.3. Dynamic extraction

Dynamic extraction is similar to free-form extraction in its flexibility, but it introduces an initial structure that can evolve as the model processes more data.

While free-form extraction allows the model to interpret and extract relationships without any predefined structure, dynamic extraction starts with a basic ontology (ie. a set of initial entity types and relationships) that guides the extraction process.

This method not only extracts relationships but also encourages the discovery of new entity types and connections, expanding the ontology as needed.

💡 **Example:** In the Panama Papers scandal, dynamic extraction might start with an initial ontology that includes entities like “Person,” “Company,” and “Offshore Account,” and relationships such as “owns” or “controls.” As the LLM processes leaked documents, it could discover new relationships like

"facilitated by" (connecting a person to a law firm) or identify new entity types such as "shell company." These discoveries would expand the original ontology, allowing the knowledge graph to capture a broader and more nuanced picture of the entities and their connections in the scandal.

How to implement dynamic extraction in practice?

```
from llama_index.core.indices.property_graph import DynamicLLMPathExtractor
from llama_index.core import PropertyGraphIndex
from llama_index.llms.openai import OpenAI

# Define the possible entity types for the knowledge graph
entities = ["PERSON", "COMPANY", "COUNTRY", "BANK", "SCANDAL"]

# Define the possible relations between the entities in the knowledge graph
relations = ["OWNS", "LOCATED_IN", "INVOLVED_IN"]

# Create an instance of SimpleLLMPathExtractor
kg_extractor = DynamicLLMPathExtractor(
    llm=OpenAI(model=LLM_MODEL, temperature=TEMPERATURE),
    allowed_entity_types=entities,
    allowed_relation_types=relations,
    )# Define the possible entity types for the knowledge graph
entities = ["PERSON", "COMPANY", "COUNTRY", "BANK", "SCANDAL"]
```

Figure 4 presents a segment of the resulting dynamic extracted graph. It demonstrates that the approach successfully identified relevant new entities and relationships, building upon the initial ones provided. This technique strikes an ideal balance between exploration and structure.

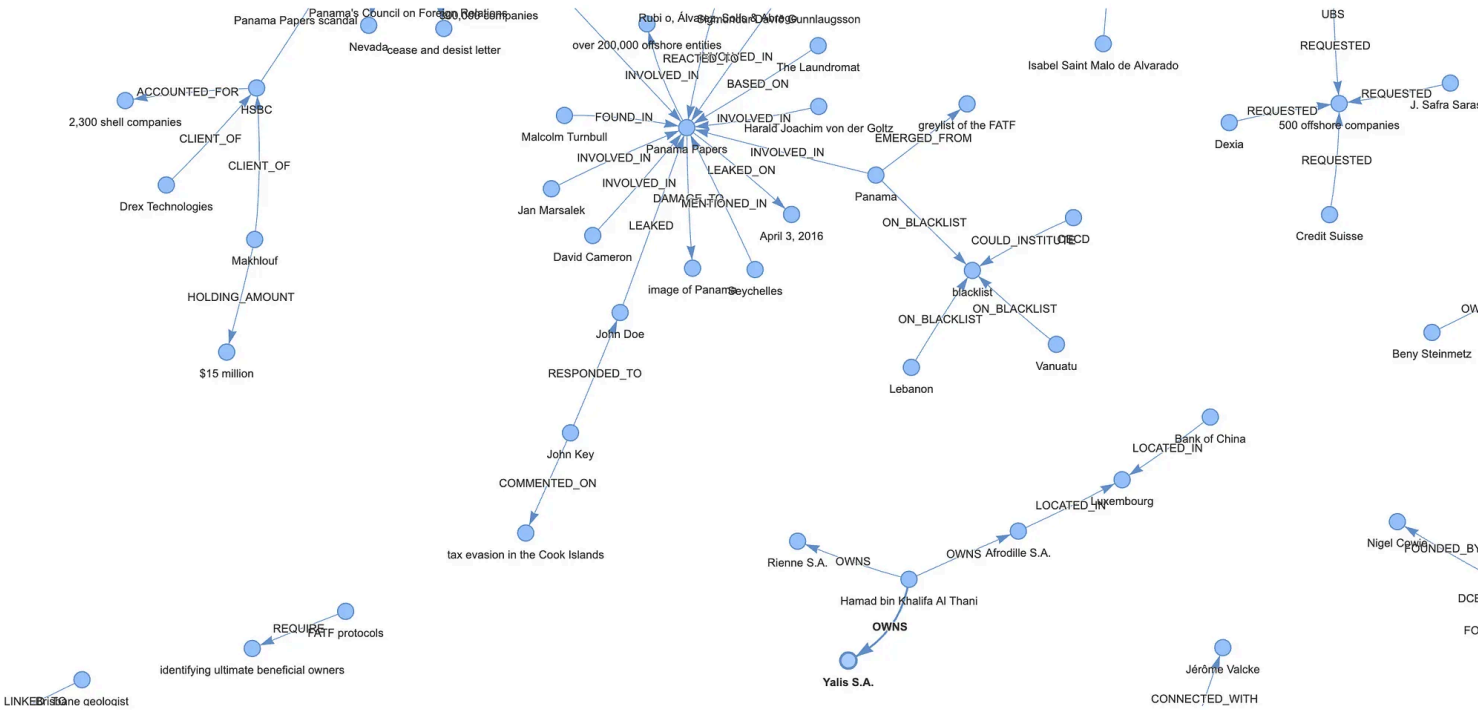


Fig 4 — Part of the dynamic extracted graph

How does the dynamic extraction method operate?

✓ **Starting with an initial ontology**: Unlike free-form extraction, which operates without a predefined schema, dynamic extraction begins with an initial ontology. This ontology includes basic entity types (e.g., “Person,” “Company”) and relationships (e.g., “owns,” “located in”). It serves as a starting point that the method can expand as it processes new data.

✓ **Guided LLM prompting**: Similar to free-form extraction, dynamic extraction uses prompts to instruct the language model (LLM) on what to extract. However, the prompts in dynamic extraction not only guide the extraction of known entities and relationships but also encourage the LLM to identify and propose new ones.

✓ **Parsing and expanding the graph**: The model's output is parsed into triples (subject-relation-object) as in free-form extraction. However, in dynamic extraction, the process includes identifying and integrating new entity types or relationships.

✓ **Ontology Evolution**: As the model processes more text, it continuously refines and expands the ontology.

2.4. Implicit extraction

The key idea of this method is to use inherent connections within the data to identify relationships between entities, even when these relationships are not explicitly mentioned. It infers connections based on the structure or context of the data, making it useful when relationships can be logically deduced rather than directly stated.

💡 **Example:** Implicit extraction is most suited when analyzing a structured document with inherent relationships, such as a legal contract or a research paper with a clear hierarchy of sections, subsections, and references. In this case, the relationships between sections (e.g., "Chapter 1 is followed by Chapter 2") or between references (e.g., "Section A cites Reference B") are not explicitly labeled as relationships but can be inferred from the document's structure. In that case, implicit extraction would automatically infer relationships like "Clause 1.1 is a subsection of Section 1" or "Clause 1.1 refers to Appendix A".

How to implement implicit extraction in practice?

Here's a quick skeleton for implementing implicit extraction in practice:

```
from llama_index.core.indices.property_graph import ImplicitPathExtractor
```

```
# Create an instance of SimpleLLMPathExtractor
kg_extractor = ImplicitPathExtractor()
```

How is implicit extraction carried out?

✓ **Node relationships:** Each node in a graph can have various types of relationships with other nodes, such as being the parent of another node, being sequentially before or after another node, or referencing another node as a source. These relationships are typically stored within the node's structure.

✓ **Edge extraction:** The extractor scans through the list of nodes and examines these inherent relationships. For each node, it identifies connections to other nodes and creates corresponding edges (relations) between them. These edges are then labeled appropriately (e.g., "PARENT," "CHILD," "NEXT," "PREVIOUS") based on the type of relationship.

✓ **Metadata integration:** During this process, the extractor can also integrate any relevant metadata associated with the nodes, such as timestamps or source information, into the relationships it creates. This enriches the graph with additional context, making the implicit connections more informative.

3. Empowering LLMs through Knowledge Graph Synergy

Once the graph is built, the challenge becomes how to effectively leverage it for LLM queries. The Llama-Index library currently offers four approaches, two of which are cypher-based.

1. Vector context retriever
2. LLM synonym retriever

3. Text-to-cypher

4. Cypher template retriever

> Cypher is a powerful query language used for interacting with graph databases, enabling precise retrieval and manipulation of graph data through pattern matching and other advanced querying techniques."

3.1. Vector context retriever

This approach retrieves nodes based on their semantic similarity to a given query, allowing for more precise and contextually appropriate search results.

💡 **Example:** Imagine searching for information related to "offshore accounts" in a property graph built from the Panama Papers data. The vector context retriever would convert the query into a vector and find nodes in the graph that are semantically similar, such as nodes representing specific companies, individuals, or transactions linked to offshore accounts. It would then retrieve these nodes, along with any directly related entities (e.g., owners or connected banks), and return them sorted by relevance.

How can the Vector Context Retriever be applied in practice?

```
# Import necessary modules from the llama_index library
from llama_index.embeddings.openai import OpenAIEmbedding
from llama_index.llms.openai import OpenAI

# Import the VectorContextRetriever class from the property_graph module
from llama_index.core.indices.property_graph import VectorContextRetriever
# Create a sub-retriever using VectorContextRetriever
# This will use the property graph store and vector store from the loaded index
# The embed_model parameter specifies the model to be used for embedding queries
sub_retriever = VectorContextRetriever(
```

```
    index.property_graph_store,
    vector_store=index.vector_store,
    embed_model=OpenAIEmbedding(model_name=EMBEDDING_MODEL),
)
# Create a retriever from the index using the previously defined sub-retriever
retriever = index.as_retriever(sub_retrievers=[sub_retriever])
# Initialize the query engine using the retriever
# The query engine will use the retriever(s) to process and return responses to
query_engine = index.as_query_engine(
    sub_retrievers=[retriever]
)
# Perform a query on the query engine to retrieve information about ICIJ's invol
print(
    query_engine.query(
    "How was the International Consortium of Investigative Journalists (ICIJ) in
    ).response
)
```

Output:

The International Consortium of Investigative Journalists (ICIJ) helped organize the research and document review once Süddeutsche Zeitung realized the scale of the work required to validate the authenticity of the leaked data. The ICIJ enlisted reporters and resources from various media outlets to investigate individuals and organizations associated with Mossack Fonseca. Additionally, the ICIJ released the leaked documents from the Panama Papers scandal on its website after verifying the source and content.

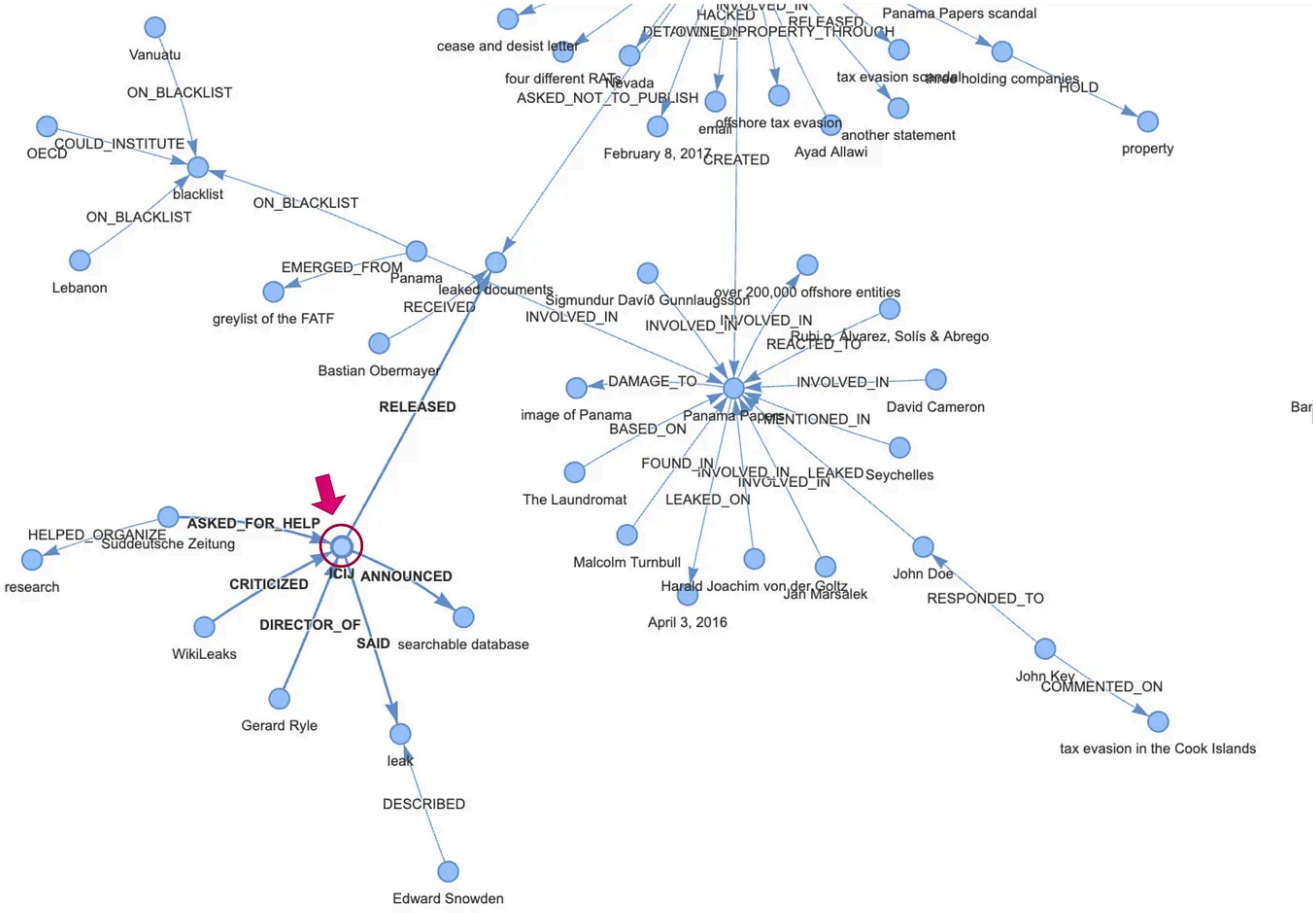


Fig 4 — Supporting part of the knowledge graph used to answer the query

How does the Vector Context Retriever retriever work?

✓ **Embedding the query**: When a query is submitted, the vector context retriever first converts the query into a vector representation using an embedding model. This vector captures the semantic meaning of the query, enabling the retriever to find similar content within the graph.

✓ **Vector store query**: The retriever identifies the nodes whose vectors are most similar to the query vector, using a similarity measure (like cosine similarity). The number of top similar nodes to retrieve is controlled by the `similarity_top_k` parameter.

✓ **Node retrieval and path expansion:** Once the most similar nodes are identified, the retriever can also explore relationships connected to these nodes based on the specified `path_depth`.

✓ **Similarity scoring:** Each retrieved node is assigned a similarity score that reflects how closely it matches the query.

✓ **Final output:** The retriever sorts the retrieved nodes by their similarity scores, optionally filters them based on the `similarity_score` threshold, and then returns the most relevant nodes and their associated relationships.

3.2. LLM synonym retriever

The LLM synonym retriever enhances the retrieval of information from a knowledge graph by expanding a user's query with synonyms generated by an LLM.

This expansion allows the retriever to capture and return more relevant nodes from the graph, improving the accuracy of the search results.

💡 **Example:** An investigator searching the Panama Papers graph for "offshore company" could use the LLM synonym retriever to expand the query with terms like "shell company" or "tax haven entity." This broader search would uncover additional relevant nodes, such as companies involved in tax evasion, that might be missed with the original query alone, leading to a more thorough investigation.

How to use the LLM synonym retriever in practice?

```
# Import necessary modules from the llama_index library
from llama_index.core.indices.property_graph import LLMSynonymRetriever
```

```
sub_retriever = LLMSynonymRetriever(
    index.property_graph_store,  # The property graph store from the index is us
    llm=OpenAI(model=LLM_MODEL, temperature=TEMPERATURE),  # Initialize the LLM
    include_text=True,  # Include the source chunk text in the retrieved paths
    max_keywords=100,  # Maximum number of keywords to be considered for retriev
    path_depth=5,  # Limit the depth of the search paths to 5 levels
)
```

What is the process behind the LLM synonym retriever?

✓ **Query expansion with synonyms:** When a query is submitted, the LLM synonym retriever first uses a language model to generate synonyms for the keywords in the query. This is done through a predefined prompt, which instructs the LLM to suggest alternative words or phrases similar to the original query terms.

✓ **Synonym integration:** The synonyms generated by the LLM are then integrated into the query. This allows the LLM to carry out a broader search within the graph.

✓ **Graph retrieval:**

- The expanded query is used to retrieve nodes from the graph. The retriever searches for nodes and relationships that match any of the terms in the expanded query.
- The depth of the retrieval, or how far the search goes in terms of relationships (e.g., direct connections or further down the relationship chain), can be adjusted using the `path_depth` parameter.

✓ **Parsing and matching:** The output from the LLM is parsed into a format that can be used to identify relevant nodes in the graph.

✓ **Node scoring and retrieval:** The matched nodes are scored based on their relevance to the original query and the relationships they hold within the graph. The retriever then returns a list of nodes with their associated scores, representing the most relevant entities and connections for the query.

3.3. Cypher-based retrievers

Llama-index offers two Cypher-based retrievers to leverage the capabilities of graph databases.

Here's a brief overview without delving into the details:

- The TextToCypherRetriever operates by leveraging a graph store schema, an input query, and a prompt template for text-to-cypher conversion to generate and execute a cypher query.
- The CypherTemplateRetriever offers a more controlled approach compared to the TextToCypherRetriever. Instead of allowing the LLM to generate any cypher statement freely, it uses a predefined cypher template, where the LLM fills in the specific details.

Key Takeaways

✓ Knowledge graphs not only improve the accuracy and contextual relevance of AI-generated knowledge but also serve as a powerful tool for industries that demand precise and interpretable information, such as healthcare, finance, and legal services.

✓ When selecting a method to build it with LLMs, it's essential to consider the trade-off between exploration and structure. Dynamic extraction offers an excellent balance for those who need both flexibility and structure in their knowledge graph.

✓ For leveraging a knowledge graph, various techniques are available, with vector-based methods providing efficient retrieval.

✓ However, it's important to acknowledge that the current version of Llama-Index has limitations in network analysis and handling complex graph structures. These capabilities would certainly improve as the synergies between knowledge graphs and LLMs are a hot topic!

References

[1] Llama index documentation

[2] Introducing the Property Graph Index: A Powerful New Way to Build Knowledge Graphs with LLMs

[3] Panama Papers

Llm | Knowledge Graph | Data Science | Genai | Deep Dives

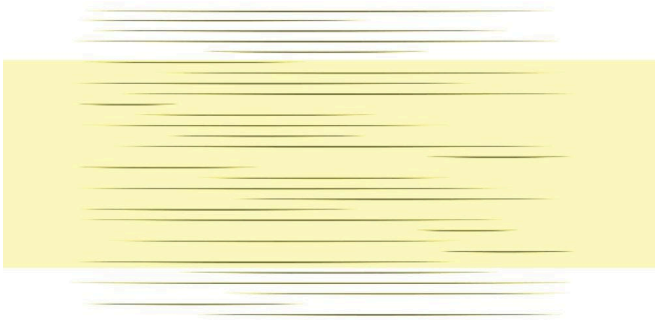
Written by Lina Faik

Follow

1.2K Followers · Writer for Towards Data Science

Senior ML Engineer & Data Scientist (Freelance) | Technical Writer | ex-Dataiku

More from Lina Faik and Towards Data Science

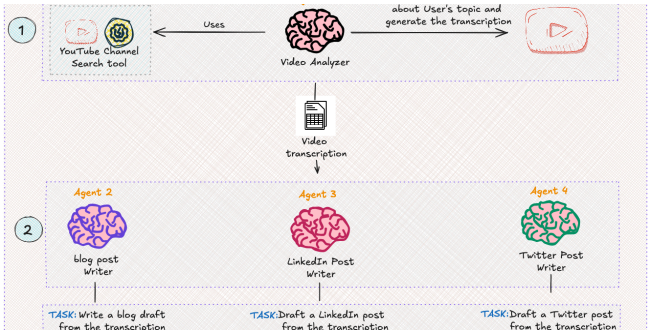


Lina Faik in Towards Data Science

Survival Analysis: Predict Time-To-Event With Machine Learning (Par...

Practical Application to Customer Churn Prediction

Feb 9, 2023 582 4



Zoumana Keita in Towards Data Science

AI Agents—From Concepts to Practical Implementation in Python

This will change the way you think about AI and its capabilities

Aug 12 1K 13

Ahmed Besbes in Towards Data Science

What Nobody Tells You About RAGs

A deep dive into why RAG doesn't always work as expected: an overview of the...

Aug 22 1.4K 20

Lina Faik in Towards Data Science

Beyond Predictions: Uplift Modeling & the Science of...

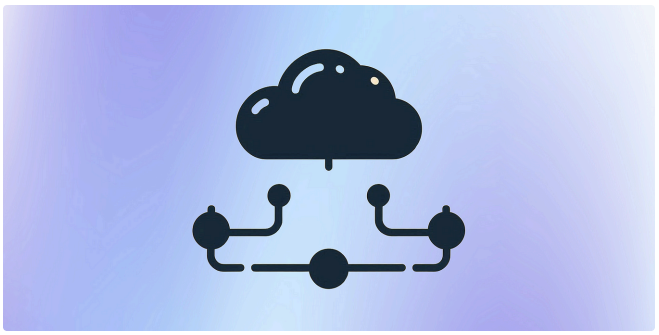
Hands-On Approach to Uplift with Tree-Based Models

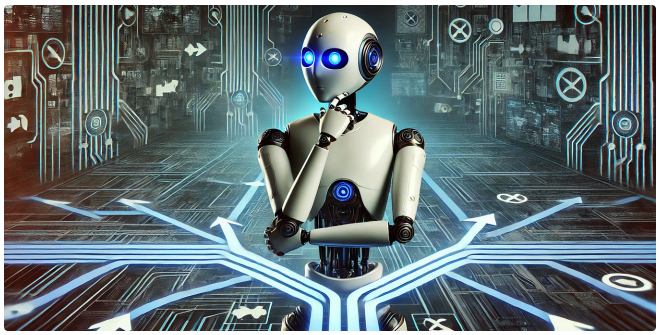
Jan 2 605 3

See all from Lina Faik

See all from Towards Data Science

Recommended from Medium





Aparna Dhinakaran in Towards Data Science

Navigating the New Types of LLM Agents and Architectures

The failure of ReAct agents gives way to a new generation of agents— and possibilities

6d ago 1K 7

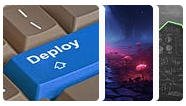
Michael Wood in Cubed

100% Accurate AI Claimed by Acurai—OpenAI and Anthropic...

Acurai's audacious claims to have discovered how LLMs operate are now confirmed by...

Aug 26 604 9

Lists



Predictive Modeling w/ Python

20 stories · 1490 saves

Natural Language Processing

1679 stories · 1252 saves

data science and AI

40 stories · 234 saves

Practical Guides to Machine Learning

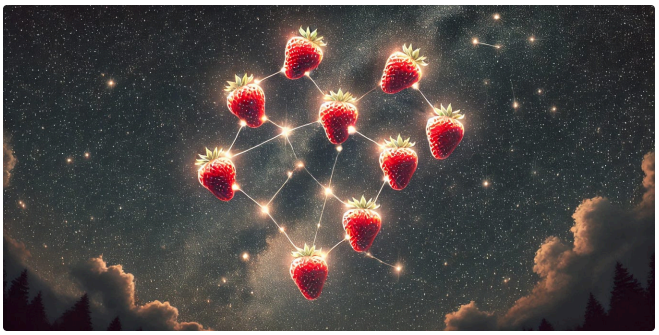
10 stories · 1819 saves

Kennedy Selvadurai, PhD in AI Advances

Local LLM Generated Knowledge Graphs Powered by Local Neo4j

How does its generation performance compare to a SimpleGraphStore storage?

5d ago 432 7

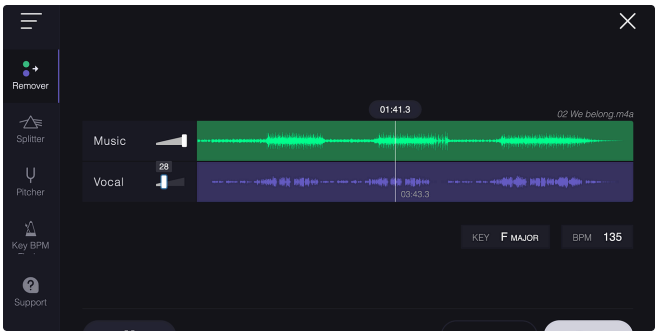


Ignacio de Gregorio

Thorough Analysis of OpenAI's New Leaked Strategy

From Fruits to Constellations

Aug 28 438 7

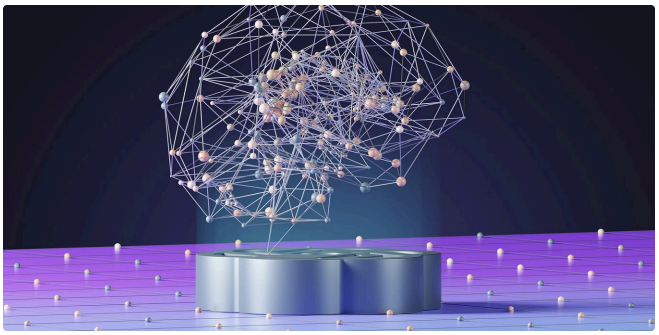


Rafe Brena, Ph.D. in Towards AI

We Are Entering The Second Wave Of Generative AI

How to tell the hype from the truth

Aug 25 160 7



Prakash Joshi Pax

Fabric: The Best AI Tool That Nobody is Talking About

An open-source AI tool to automate every day tasks

Aug 25 395 3

See more recommendations