Independent Study Report

Identifying Media Bias Using GraphRAG and Neo4j

Table of Contents

1. Executive Summary

This project investigates the detection and analysis of media bias using advanced Natural Language Processing (NLP) techniques, focusing on the comparative performance of Retrieval-Augmented Generation (RAG) and Graph-based Retrieval-Augmented Generation (GraphRAG) systems. The central goal is to improve transparency and accountability in journalism by enabling contextual, relationship-aware exploration of news narratives.
To support this goal, we constructed a custom dataset of 60 politically and non-politically themed news articles sourced from a diverse set of outlets including CNN, Fox News, HuffPost, and Reuters. We employed Named Entity Recognition (NER) using spaCy to extract PERSON entities and their co-occurrence relationships, then used Neo4j to build a semantic knowledge graph linking entities across articles. This graph structure enriched the context available for answering complex, relationship-oriented questions.

Evaluation was conducted using RAGAS metrics- Relevancy, Faithfulness, Precision, and Recall - across a variety of question types. Results demonstrated that while RAG performed better on fact-based summaries, GraphRAG significantly outperformed on queries involving relational context and cross-document connections, such as identifying framing biases or entity co-occurrence patterns. This research validates the value of hybrid systems that combine symbolic reasoning (graphs) with neural embeddings to support interpretable, high-fidelity responses. The proposed GraphRAG pipeline enables new ways to audit media framing, trace entity narratives across sources, and ask semantically rich questions about news content, offering practical applications for media researchers, watchdog organizations, and public policy analysts.

2. Problem Definition & Significance

With the overwhelming volume of news and media published daily often with varying perspectives, it's becoming increasingly important to accurately summarize content and answer complex questions about it. While large language models (LLMs) have made impressive progress in natural language understanding, they can struggle when a question depends on specific context or deeper relationships between people, events, or sources.
Retrieval-Augmented Generation (RAG) has helped by allowing LLMs to pull in relevant documents, but this method mostly relies on surface-level similarity. As a result, it can miss the bigger picture - especially when dealing with questions that involve how different entities relate to each other or how narratives develop across stories. This is where Graph-based RAG (GraphRAG) comes in: by combining traditional retrieval with knowledge graphs, it adds a more structured and connected understanding of the content.

In this project, we explore when and why GraphRAG can offer advantages over traditional RAG. We built a knowledge graph from real news articles and ran both pipelines to evaluate how they perform on different types of questions. This matters not only for advancing retrieval methods, but also for practical uses in journalism, research, and media monitoring - where context and relationships often matter just as much as facts.

3. Prior Literature

Author(s)	Use Case	Data Source	Technology	Key Findings
Zahra et al. (2024)	Fetching soccer stats efficiently using natural language	Soccernet	Structured-GraphRAG, Knowledge Graphs, Cypher queries	97% faster than traditional search, avoids hallucination, uses 4-stage pipeline for accurate stat retrieval
Junde et al. (2024)	Evidence-based medical question answering using graph-enhanced RAG (MedGraphRAG)	MIMIC-IV, MedC-K, fake/public health claims, UMLS graph	MedGraphRAG, Triple Graph Construction, U-Retrieval, Meta-MedGraph Tagging	Higher citation precision, recall, and better understandability; tested with GPT-4, Gemini-Pro, Llama2/3
Wei & Rongsheng (2024)	Efficient speech recognition using adaptive graph-based model compression (AMDP-GraphRAG)	LibriSpeech, TED-LIUM Release 3	GNNs, Adaptive Multilevel Distillation, knowledge distillation, model pruning	Reduced WER, lower memory usage, improved generalization, and better efficiency for edge devices
Tenteng et al. (2021)	Predicting customer engagement (likes/comments/shares) on Facebook brand posts	Facebook data from 7 car brands (2012–2016)	Meta-path based HIN, Graph Neural Networks, attention mechanism	Outperformed baselines; attention provided interpretability; emphasized network structure and content
Darren et al. (2025)	Answering global sensemaking questions over long corpora (e.g., podcasts, news)	Podcast transcripts, news articles	GraphRAG, LLMs, Leiden community detection, hierarchical summaries, map-reduce	Outperformed vector RAG in answer quality and diversity; efficient token usage; suitable for large-scale summarization and reasoning
Zirui et al. (2024)	Answering complex domain-specific queries via LightRAG	UltraDomain benchmark datasets (Agriculture, CS, Legal, Mix)	LightRAG, graph-based text indexing, dual-level retrieval, entity/relationship extraction, vector indexing	Higher retrieval accuracy, faster than traditional RAGs on large datasets, better adaptability and cost-efficiency

4. Data source & Preparation

As the volume of digital news continues to grow, so does the challenge of accurately understanding, summarizing, and extracting insights from complex media narratives. Large Language Models (LLMs) like GPT have proven capable in many natural language tasks, but they often struggle with queries that require contextual depth or relational understanding, particularly when multiple people, events, or entities are involved.

Retrieval-Augmented Generation (RAG) improves this by allowing models to refer to relevant documents before generating a response. However, it still treats text as unstructured data, limiting its ability to handle relationship-heavy questions. Graph-based RAG (GraphRAG) addresses this gap by retrieving context from structured knowledge graphs, allowing models to better understand connections between entities.

To evaluate both methods, we curated a custom dataset of 60 news articles spanning topics such as tariffs, the Trump-Zelenskyy meeting, California wildfires, and NCAA basketball. Articles were scraped from diverse media sources like CNN, Fox News, HuffPost, and Reuters. We then preprocessed the content using Named Entity Recognition (NER) via spaCy to extract PERSON entities and inferred relationships based on co-occurrence in sentences. These were used to build a knowledge graph in Neo4j.

This setup allowed us to test both RAG and GraphRAG pipelines across different question types fact-based, summary-oriented, and relationship-driven. By comparing performance using RAGAS metrics, we aimed to understand the practical significance of structured versus unstructured retrieval in real-world media analysis tasks.

5. Methodology & Results

To explore how knowledge graphs can enhance retrieval-augmented generation, we built a GraphRAG pipeline that combines the structure of Neo4j with the reasoning power of GPT-4o. This setup allowed for context-rich responses grounded in real news articles, linked through relationships between people mentioned in the news.

i). Data Collection
We selected 60 news articles from CNN, Fox News, HuffPost, Reuters, and NPR, covering four diverse topics:

- U.S. tariffs
- Russia Ukraine War (Trump-Zelenskyy meeting)
- California wildfires

- NCAA Basketball championship

Articles were selected to capture a wide range of language, complexity, and named entities, particularly people. Notably, two of the topics Trump–Zelenskyy meeting and U.S. tariffs showed clear political bias across sources, while the other two California wildfires and NCAA Basketball championship were largely factual and non-partisan in coverage.

Media Bias Chart

To understand and analyze media bias, we used the AllSides Media Bias Chart, which classifies news outlets based on their political leanings. The chart divides media sources into five categories:

- Extreme Left: Strongly liberal or progressive with a clear partisan tone.
- Left-Leaning: Generally progressive, focusing on social issues and equity.
- Center: Balanced or mixed reporting that avoids partisan bias.
- Right-Leaning: Generally conservative, emphasizing traditional values and market freedom.
- Extreme Right: Strongly conservative with a highly partisan or nationalist perspective.

This classification helped us group the articles and evaluate how language and framing differed across the political spectrum.

All sides media bias chart:

ii). Preprocessing and Entity Extraction

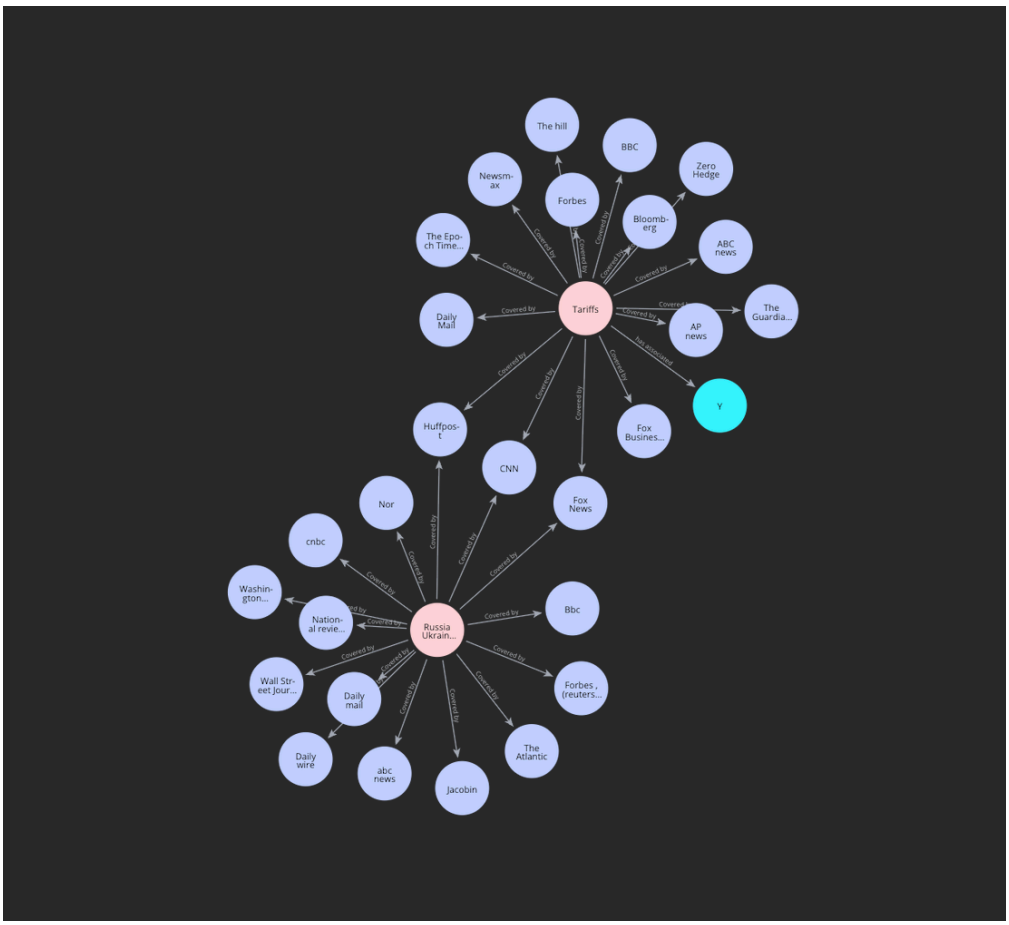
We used spaCy for Named Entity Recognition (NER), extracting all entities tagged as PERSON. Once extracted, we scanned each sentence for co-occurring persons. If two or more people appeared in the same sentence, we assumed a potential relationship and created undirected edges between them. These entity pairs were used to build a lightweight person-centric knowledge graph.

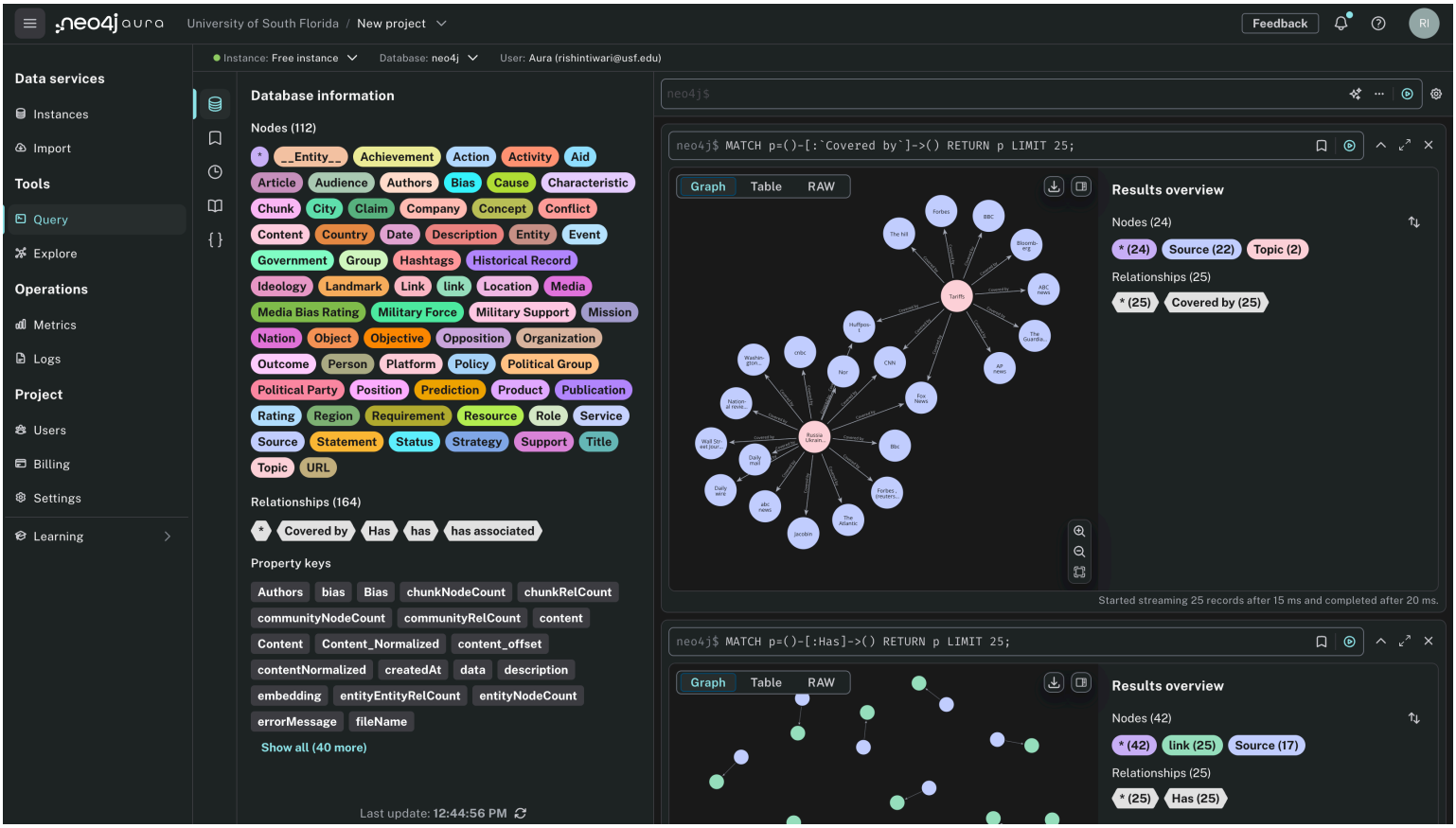
iii). Knowledge Graph Construction with Neo4j

The extracted entities and relationships were stored in a Neo4j graph database instance, which provided a robust and queryable backend for our retrieval process.

- Nodes: Each person was stored as a node with properties such as name, article_title, and a list of source_sentences where the person appeared.
- Edges: Relationships were stored as undirected edges with attributes such as the sentence text and source article.
- The graph was constructed programmatically using the neo4j Python driver and stored on a Neo4j instance.

This structure allowed us to move beyond flat document chunks and retrieve semantically linked entities for graph-aware retrieval.





iv). Neo4j LLM KG Builder & GPT-4o Integration

To enable natural language interactions with the graph, we used Neo4j's LLM KG Builder, which provides a low-code interface to connect LLMs like GPT-4o with Neo4j's underlying data. Key capabilities we implemented:

- Graph Query Routing: User questions are parsed and interpreted using GPT-4o to determine relevant subgraphs (e.g., "Show connections between Trump and Zelenskyy").
- Cypher Generation: GPT-4o, via the LLM Builder, generates Cypher queries dynamically and executes them against Neo4j.
- Conversational QA: Responses from Cypher queries are interpreted and reformulated by GPT-4o into coherent natural language answers.

We also used Neo4j's built-in vector index and the KnowledgeGraphIndex for hybrid retrieval in the GraphRAG pipeline. The graph-backed context was then passed to GPT-4o for response generation.

v). Retrieval-Augmented Generation with GPT-4o

We employed GPT-4o for two main purposes:

- Question Understanding & Graph Navigation: GPT-4o interprets the question and determines which nodes/relationships are most relevant.

- Grounded Response Generation: It receives the extracted context (from Neo4j or the KnowledgeGraphIndex) and generates a fluent, contextually accurate response.

The final prompt fed into GPT-4o includes:

- The question / prompt
- A set of relevant nodes and edges from the graph
- Sentences from source articles containing those entities

vi). Evaluation with RAGAS

We evaluated the generated responses using RAGAS, focusing on:

- Context Recall - Completeness of the retrieved information
- Faithfulness - Accuracy of generated answers with respect to the source context
- Answer Relevance - How directly the answer addressed the question

$$\text{Faithfulness} = \frac{\text{Number of Correct Facts in the Response}}{\text{Total Number of Facts in the Response}}$$

$$\text{Answer Relevancy} = \frac{\text{Number of Relevant Concepts in the Response}}{\text{Total Number of Concepts in the Response}}$$

$$\text{Context Precision} = \frac{\text{Number of Relevant Sentences Retrieved}}{\text{Total Number of Sentences Retrieved}}$$

Results:

Prompt: Generate a summary of 150 words (NCAA championship, Trump–Zelenskyy meeting, etc.)

	Gpt-4o	Gpt-4o + RAG	Gpt-4o + Graph RAG
NCAA championship	Relevancy: 0.73 Context Recall: 0.70 Faithfulness: 0.60	Relevancy: 0.82 Context Recall: 0.58 Faithfulness: 0.65	Relevancy: 0.74 Context Recall:0.70 Faithfulness: 0.75
California wildfire	Relevancy: 0.72 Context Recall: 0.33 Faithfulness: 0.67	Relevancy: 0.73 Context Recall: 0.37 Faithfulness: 0.33	Relevancy: 0.73 Context Recall:0.42 Faithfulness: 0.44

Russa-Ukraine war	Relevancy: 0.76 Context Recall: 0.46 Faithfulness: 0.82	Relevancy: 0.76 Context Recall: 0.24 Faithfulness: 0.81	Relevancy: 0.76 Context Recall:0.65 Faithfulness: 1.00
US tariffs	Relevancy: 0.80 Context Recall: 0.38 Faithfulness: 0.81	Relevancy: 0.80 Context Recall: 0.36 Faithfulness: 0.69	Relevancy: 0.81 Context Recall:0.58 Faithfulness: 0.68

Prompt: Generate an extreme left / left-leaning / centrist / right-leaning / extreme right summary of 150 words (NCAA championship, Trump–Zelenskyy meeting, etc.)

	Gpt-4o	Gpt-4o + RAG	Gpt-4o + Graph RAG
NCAA championship	Relevancy: 0.83 Context Recall: 0.13 Faithfulness: 0.0	Relevancy: 0.82 Context Recall: 0.13 Faithfulness: 0.04	Relevancy: 0.72 Context Recall:0.57 Faithfulness: 0.27
California wildfire	Relevancy: 0.79 Context Recall: 0.25 Faithfulness: 0.47	Relevancy: 0.86 Context Recall: 0.04 Faithfulness: 0.11	Relevancy: 0.78 Context Recall:0.24 Faithfulness: 0.34
Russa-Ukraine war	Relevancy: 0.81 Context Recall: 0.44 Faithfulness: 0.40	Relevancy: 0.77 Context Recall: 0.36 Faithfulness: 0.39	Relevancy: 0.77 Context Recall:0.20 Faithfulness: 0.29
US tariffs	Relevancy: 0.82 Context Recall: 0.15 Faithfulness: 0.23	Relevancy: 0.84 Context Recall: 0.14 Faithfulness: 0.11	Relevancy: 0.76 Context Recall:0.07 Faithfulness: 0.08

GraphRAG Generally Improves Faithfulness:

- Russia-Ukraine war: GraphRAG achieved the highest faithfulness (1.00) and strong context recall (0.65), outperforming the other two methods.
- NCAA Championship: GraphRAG also had the best context recall (0.70, 0.57) and faithfulness (up to 0.75), suggesting better summarization from structured context.

RAG Shows Strength in Relevancy:

- RAG consistently delivered high relevancy scores, especially for US tariffs (0.80–0.84) and California wildfires (up to 0.86), though often with lower context recall and faithfulness.

Fulltext Baseline (GPT-4o without retrieval) Struggles on Context:

- Faithfulness and recall were especially low (even 0.0) for some topics like NCAA, showing the value of augmentation.

We also experimented with Ollama running LLaMA 3.1 locally to query the graph database via a custom RAG pipeline. This enabled privacy-friendly and offline access to the same knowledge base, leveraging the graph structure to generate context-rich answers. LLaMA 3.1, paired with embedding-based retrieval from the graph, offered a lightweight yet effective alternative for querying the structured information, especially for developer-controlled environments.

6. Insights & Recommendations

Our exploration using the GraphRAG pipeline revealed that incorporating a knowledge graph (KG) significantly enhances retrieval-augmented generation tasks by structuring facts and relationships from unstructured news articles. By extracting entities and linking them contextually in a Neo4j graph database, we enabled GPT-4o to ground its responses with more accuracy and relevance. The Neo4j LLM GraphRAG Builder allowed us to seamlessly integrate GPT-4o's conversational capabilities with the KG, enabling dynamic, context-aware querying through natural language. This not only improved the coherence and factual grounding of summaries but also enabled deep drill-downs based on political bias, topic, or named entities. We recommend leveraging this hybrid approach in any domain where factual consistency, traceability, or personalized retrieval is critical. GraphRAG is particularly beneficial for media monitoring, misinformation detection, knowledge curation, and sentiment-aware reporting. Future work could extend this with more detailed datasets and exploring the KG and GraphRAG capabilities in different domains.

7. References

https://arxiv.org/html/2409.17580v1

https://arxiv.org/pdf/2408.04187

http://www.jestr.org/downloads/Volume17Issue6/fulltext11762024.pdf

https://scholarspace.manoa.hawaii.edu/server/api/core/bitstreams/f619b4c7-a55b-4e3b-acff-d844f9408ca7/content

https://arxiv.org/pdf/2404.16130

https://arxiv.org/html/2410.05779v1

https://microsoft.github.io/graphrag/

https://neo4j.com/blog/genai/graphrag-manifesto/

https://neo4j.com/blog/developer/graphrag-llm-knowledge-graph-builder/

https://www.allsides.com/media-bias/media-bias-chart

8. Appendix

Scraping code :

```
import pandas as pd
import requests
from newspaper import Article
from tqdm import tqdm
import time

# Step 1: Load table
df = pd.read_csv('dataset.csv')  # or use read_excel

# Headers to mimic a real browser
HEADERS = {
   "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) "
                 "AppleWebKit/537.36 (KHTML, like Gecko) "
                 "Chrome/122.0.0.0 Safari/537.36"
}

# Step 2: Scrape articles
df['Content'] = ''

for idx, row in tqdm(df.iterrows(), total=len(df)):
   url = row['Link']
   try:
      # Make request manually with headers
      response = requests.get(url, headers=HEADERS, timeout=10)
      if response.status_code != 200:
         print(f"Failed to access {url}: Status code {response.status_code}")
         df.at[idx, 'Content'] = ''
         continue

      # Pass HTML manually to newspaper
      article = Article(url)
      article.set_html(response.text)
      article.parse()
```

```
        df.at[idx, 'Content'] = article.text

    except Exception as e:
        print(f"Failed to scrape {url}: {e}")
        df.at[idx, 'Content'] = ''

    time.sleep(1)  # polite delay between requests

# Step 3: Save table
df.to_csv('updated_table_with_content.csv', index=False)
print("✅ Scraping complete and file saved as 'updated_table_with_content.csv'")
```

NER code

```
import pandas as pd
import spacy
import re
from tqdm import tqdm
from collections import defaultdict, Counter

# Load Data and spaCy model
print("Loading data and model...")
df = pd.read_csv('updated_table_with_content.csv')
nlp = spacy.load('en_core_web_sm')

# Step 1: Extract Named Entities (PERSON)
def extract_named_entities(texts):
    persons = []
    print("Extracting named entities...")
    for doc in tqdm(nlp.pipe(texts, batch_size=20), total=len(texts)):
        persons.extend([ent.text.strip() for ent in doc.ents if ent.label_ == "PERSON"])
    return persons

all_persons = extract_named_entities(df['Content'].dropna().tolist())
print(f"Total PERSON entities extracted: {len(all_persons)}")

# Step 2: Clean and Filter NEs
def clean_persons(persons):
    cleaned = []
    for p in persons:
```

```
        if len(p.split()) <= 3 and all(word[0].isupper() for word in p.split() if word.isalpha()):
            cleaned.append(p)
    return cleaned

filtered_persons = clean_persons(all_persons)
print(f"PERSON entities after cleaning: {len(filtered_persons)}")

# Step 3: Build Normalization Dictionary (Smarter Grouping)
def build_normalization_dict(person_list):
    groups = defaultdict(list)

    for person in person_list:
        parts = person.replace(".", "").split()
        if parts:
            last_name = parts[-1].lower()
            groups[last_name].append(person)

    normalization_dict = {}
    for group, names in groups.items():
        name_counter = Counter(names)
        canonical_name = max(name_counter, key=name_counter.get)  # most frequent
        for name in names:
            if name != canonical_name:
                normalization_dict[name] = canonical_name

    return normalization_dict

normalization_dict = build_normalization_dict(filtered_persons)
print(f"Normalization dictionary size: {len(normalization_dict)}")

# Optional: See top mapped examples
print("\nSample normalization mappings:")
for i, (k, v) in enumerate(normalization_dict.items()):
    if i >= 10:
        break
    print(f"  {k}  -->  {v}")

# Step 4: Normalize Content
def normalize_content(text, normalization_dict):
    if pd.isna(text) or text.strip() == '':
        return text
    for pattern, replacement in normalization_dict.items():
        text = re.sub(rf'\b{re.escape(pattern)}\b', replacement, text, flags=re.IGNORECASE)
```

```
    return text

print("\nNormalizing articles...")
tqdm.pandas()
df['Content_Normalized'] = df['Content'].progress_apply(lambda x: normalize_content(x,
normalization_dict))

# Step 5: Save the final file
df.to_csv('final_normalized_articles.csv', index=False)
print("\n✅ Final normalized articles saved as 'final_normalized_articles.csv'")
```

Neo4j code using llama3.1

```
from neo4j import GraphDatabase
from neo4j_graphrag.retrievers import VectorRetriever
from neo4j_graphrag.generation import GraphRAG
from langchain_community.chat_models import ChatOllama
from sentence_transformers import SentenceTransformer
from langchain.embeddings import SentenceTransformerEmbeddings

# 1. Neo4j connection
URI = "neo4j+s://da0d5023.databases.neo4j.io"
AUTH = ("neo4j", "")
INDEX_NAME = "vector"

# 2. Connect to Neo4j
driver = GraphDatabase.driver(URI, auth=AUTH)

# 3. Embedder - using sentence-transformers (384 dim)
embedder = SentenceTransformerEmbeddings(model_name="all-MiniLM-L6-v2")

# 4. Retriever
retriever = VectorRetriever(driver, INDEX_NAME, embedder)

# 5. Local LLM via Ollama
llm = ChatOllama(model="llama3.1:8b-instruct-q8_0")

# 6. GraphRAG
rag = GraphRAG(retriever=retriever, llm=llm)

# 7. Ask the question
```

```
query_text = "How did the meeting went give 5 summaries for the 5 different document?"
response = rag.search(query_text=query_text, retriever_config={"top_k": 3})

# 8. Output
print(response.answer)
```