# Mastering SERP Analysis & Knowledge Graphs

Elias Dabbas, featuring Andrea Volpini from WordLift and Ray Grieselhuber from DemandSphere.

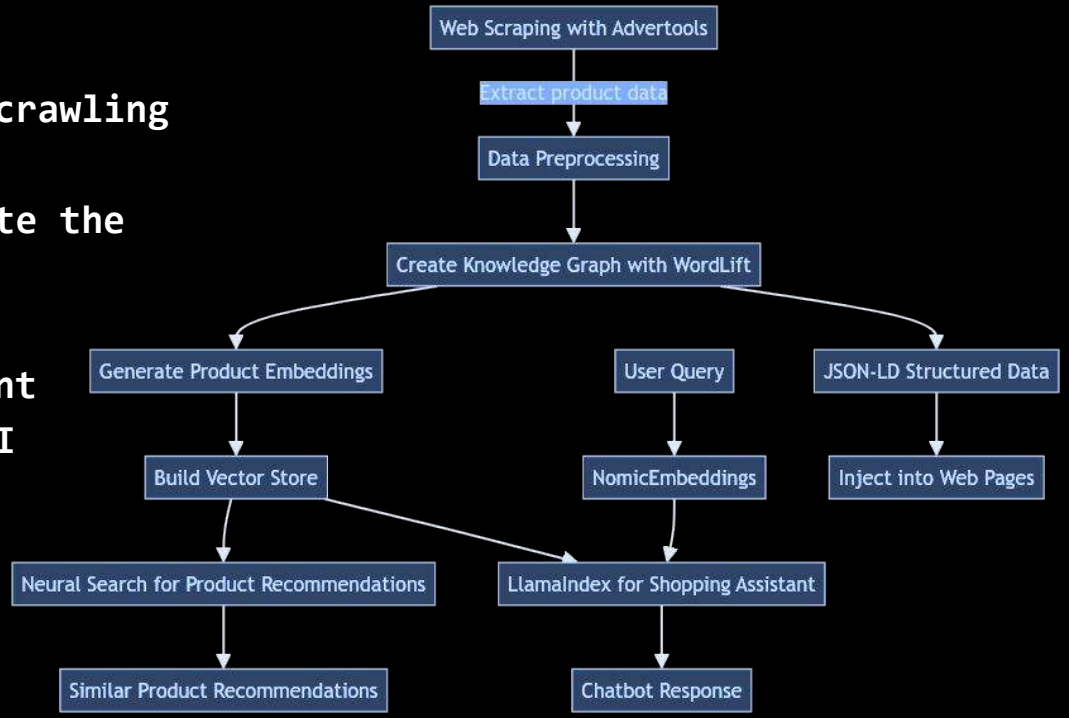
Jul 2024


# Can we build a *Product* Knowledge Graph?

Using Crawl Data From Advertools

# Meet our Friends: Essential Libraries for today

- `advertools`: For efficient web crawling and scraping
- `RDFLib`: To create and manipulate the graph from scraped data
- `WordLift`: Our main tool for KG optimization and SEO enhancement
- `Llama Index`: For building an AI Assistant using our KG



# WordLift Client and KG Creation

```
pip install
wordlift-client
```

Setting up WordLift client and creating entities.

```python
from wordlift_client.apis import EntitiesApi
from rdflib import Graph, Namespace
import json

# WordLift API configuration
API_KEY = 'YOUR_API_KEY'  # Replace with your actual API key
BASE_URI = 'YOUR_DATASET_BASE_URI'

# Define custom namespace
EXAMPLE_PRIVATE_NS = Namespace("https://ns.example.org/private/")

async def create_entity(entities_api, entity_data):
    g = Graph().parse(data=json.dumps(entity_data), format='json-ld')
    body = g.serialize(format='application/rdf+xml')
    await entities_api.create_or_update_entities(
        body=body,
        _content_type='application/rdf+xml'
    )

async def build_knowledge_graph(df, dataset_uri, api_client):
    entities_api = EntitiesApi(api_client)
    for _, row in df.iterrows():
        if row['page_type'] == 'PDP':
            entity_data = create_product_entity(row, dataset_uri)
        elif row['page_type'] == 'PLP':
            entity_data = create_collection_entity(row, dataset_uri)
        await create_entity(entities_api, entity_data)
```

# Entity Creation & Embeddings

**Creating entities with embedding generation. All in one step!**

```python
def create_product_entity(row, dataset_uri):
    url = replace_url(row['url'])
    product_entity_uri = create_entity_uri(url)

    entity_data = {
        "@context": "http://schema.org",
        "@type": "Product",
        "@id": product_entity_uri,
        "url": url,
        "name": row['title'] if not pd.isna(row['title']) else "Untitled Product",
        "urn:meta:requestEmbeddings": [
            "http://schema.org/name",
            "http://schema.org/description"
        ],
    }

    if not pd.isna(row.get('product_description')):
        entity_data["description"] = row['product_description']

    # Add price information if available
    if not pd.isna(row.get('product_price')):
        price = clean_price(row['product_price'])
        if price is not None:
            offer_entity_uri = f"{product_entity_uri}/offer_1"
            entity_data["offers"] = {
                "@type": "Offer",
                "@id": offer_entity_uri,
                "price": str(price),
                "priceCurrency": "GBP",
                "availability": "http://schema.org/InStock",
                "url": url
            }

    return entity_data
```

"urn:meta:requestEmbeddings" field tells WordLift which properties to use for generating embeddings.

NOMIC

# Query Product Data via GraphQL

Extract only the information we need with a simple GraphQL query.

```python
async def perform_graphql_query(api_client):
    graphql_api = GraphQLApi(api_client)
    query = """
    {
        products(rows: 10) {
            id: iri
            category: string(name:"schema:category")
            name: string(name:"schema:name")
            description: string(name:"schema:description")
            url: string(name:"schema:url")
        }
    }
    """
    request = GraphqlRequest(query=query)

    try:
        response = await graphql_api.graphql_using_post(body=request)
        print("GraphQL Query Results:")
        print(json.dumps(response, indent=2))
    except Exception as e:
        logger.error(f"An error occurred during GraphQL query: {e}")

async with ApiClient(configuration) as api_client:
 # Step 6: Perform GraphQL query
 await perform_graphql_query(api_client)
 logger.info("Knowledge graph building and GraphQL query completed.")
```

WordLift

# Can we automate *Structured Data*?

schema.org

# WordLift Data API

This API is used to inject structured data markup from the Knowledge Graph (KG) into your webpages.

We are referencing a fictitious URL: https://data-science-with-python-for-seo.wordlift.dev.

When calling WordLift's data API, we simply pass a URL and receive the corresponding JSON-LD (JavaScript Object Notation for Linked Data).

```python
def get_json_ld_from_url(url):
    # Construct the API URL by prefixing with 'https://api.wordlift.io/data/https/'
    api_url = 'https://api.wordlift.io/data/https/' + url.replace('https://', '')

    # Make the GET request to the API
    response = requests.get(api_url)

    # Check if the request was successful
    if response.status_code == 200:
        # Parse the JSON-LD from the response
        json_ld = response.json()
        return json_ld
    else:
        print(f"Failed to retrieve data: {response.status_code}")
        return None

def pretty_print_json(json_obj):
    # Pretty print the JSON object
    print(json.dumps(json_obj, indent=4))
```

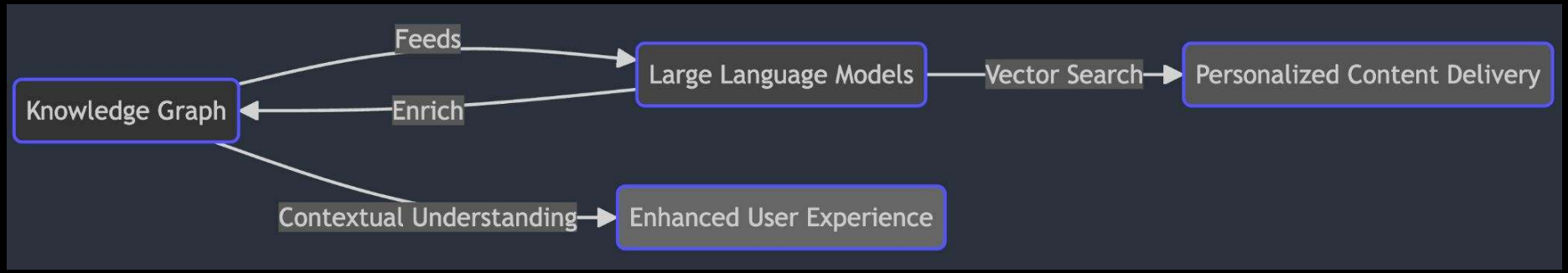
schema.org

WordLift

# Can we run *Neural Search*?

Product Recommendation and
Internal Links

# **Neuro-Symbolic AI:** DISCRETE AND CONTINUOUS KR BECOME INTEROPERABLE




The interplay between the Knowledge Graph (*discrete*), Large Language Models and vector search (*continuous*) for an **improved user interaction, better findability** (SEO) and **personalized content delivery.**

# Finding Similar Products

Using Hierarchical Navigable Small Worlds (HNSW) algo we can deliver efficient vector search using data in the Knowledge Graph

```python
async def get_top_k_similar_urls(configuration, query_url: str, top_k: int):
    request = VectorSearchQueryRequest(
        query_url=query_url,
        similarity_top_k=top_k,
    )

    async with wordlift_client.ApiClient(configuration) as api_client:
        api_instance = VectorSearchQueriesApi(api_client)
        try:
            page = await api_instance.create_query(vector_search_query_request=request)
            return [
                {
                    "url": item.id,
                    "name": item.text.split('\n')[0],
                    "score": item.score
                }
                for item in page.items if item.id and item.text
            ]
        except Exception as e:
            logger.error(f"Error querying for entities: {e}", exc_info=True)
            return None
```

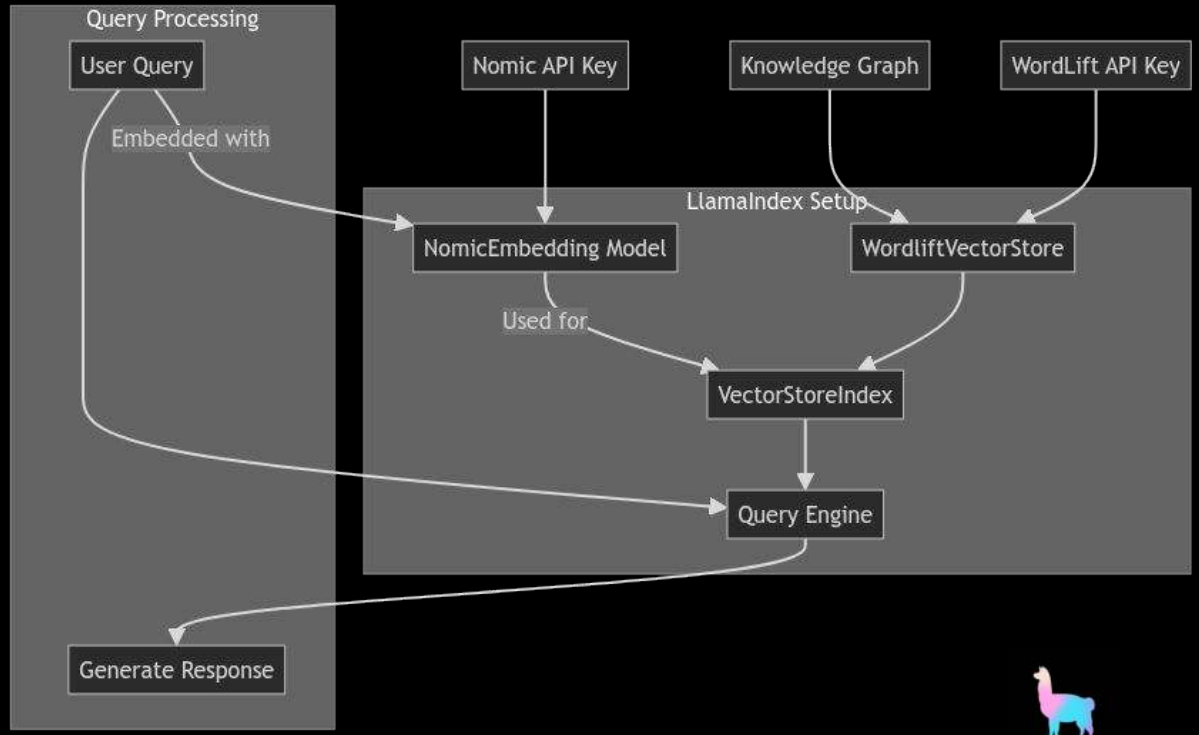
WordLift

# Can we build an *AI Agent*?

Using Llama Index

# One-Two-Three: Graph-RAG!

- **Knowledge Graph**: The pre-existing Knowledge Graph is the starting point.
- **WordliftVectorStore**: The Knowledge Graph is made accessible as a WordliftVectorStore.
- **NomicEmbedding Model**: A NomicEmbedding model is set up using the Nomic API key.
- **VectorStoreIndex**: A VectorStoreIndex is created from the WordliftVectorStore, utilizing the NomicEmbedding model.
- **Query Engine**: A Query Engine is created from the VectorStoreIndex.



WordLift

# Are you ready?

https://wor.ai/free-trial

FREE TRIAL