AI-Based Semantic Knowledge Graph Construction and Visualization from Text Data

Done by:

SaiTejaswi

Jahnavi Lakshmi

Poobesh Raj

Shivam Kanse

SYNOPSIS

The **Semantic Knowledge Graph Generator** is a project aimed at transforming unstructured textual data into structured, interconnected knowledge representations. With the exponential growth of textual information in research papers, articles, and reports, there is a critical need for systems that can extract meaningful entities, detect relationships, and visualize these connections effectively.

This project utilizes **Natural Language Processing (NLP)** techniques, including **Named Entity Recognition (NER)** and **relation extraction**, to identify entities such as persons, organizations, and concepts, and establish meaningful links between them. The system further employs **semantic similarity models**, specifically **SentenceTransformer embeddings**, to link related concepts across multiple datasets, enabling cross-domain knowledge integration.

The extracted entities and relationships are represented as a **knowledge graph**, constructed using **NetworkX** for graph analytics and **PyVis** for interactive visualization. Users can explore the graph dynamically, perform semantic searches, ask natural language questions, and receive relevant answers from the graph. This approach allows intuitive exploration and discovery of knowledge that is otherwise hidden in unstructured text.

To enhance usability, the system is deployed as a **web-based application using Streamlit**, providing an interactive interface for uploading datasets, visualizing knowledge graphs, and querying the system in real time. The project demonstrates the effectiveness of combining NLP, graph theory, and semantic embeddings to create an automated, user-friendly platform for knowledge extraction and exploration.

CHAPTER I

INTRODUCTION

- In the era of digital transformation, the volume of unstructured data has increased exponentially. Extracting meaningful insights from this data has become an essential challenge in the fields of artificial intelligence and data science. Traditional keyword-based search and information retrieval techniques are limited in their ability to understand the contextual relationships between concepts. To overcome these limitations, knowledge graphs have emerged as a powerful framework to represent information as a network of entities and their interconnections.

- The Semantic Knowledge Graph Generator aims to automate the process of building knowledge graphs from textual data. It integrates Natural Language Processing (NLP) tools like SpaCy for Named Entity Recognition (NER) and relation extraction, and employs semantic similarity models such as Sentence Transformers for linking related concepts across domains. The project further includes modules for visualization, graph analytics, and semantic querying, enabling users to interactively explore the generated knowledge base.

- By incorporating semantic embeddings and machine learning models, this system not only identifies relationships but also measures their contextual similarity. The application supports tasks such as semantic search, domain linking, and question answering, demonstrating the utility of knowledge graphs in intelligent data exploration. Ultimately, this project contributes to making large-scale text data more interpretable and actionable for research and enterprise use cases.

CHAPTER II

PROBLEM STATEMENT

2.1 Problem Statement:

Unstructured textual data, such as articles, research papers, and reports, often contains valuable information that is difficult to utilize effectively due to the lack of structure and semantic connections. Traditional text processing techniques, including keyword searches or basic parsing, fail to identify meaningful relationships and are unable to support cross-domain knowledge retrieval.

2.1.1 Understand the Problem Statement:

There is a growing need for an **automated system** that can:

1. Extract entities and detect relationships from unstructured text.
2. Represent these entities and relationships in a structured, visual, and query able form.
3. Enable semantic search and question answering to retrieve meaningful information.
4. Link related concepts across multiple datasets for cross-domain knowledge integration.

2.1.2 Goal of the Problem Statement:

The primary goal of this project is to transform unstructured textual data into a **semantic knowledge graph**, facilitating better information discovery, reasoning, and practical insights for research, academics, and enterprise applications.

2.1.3 Objectives of the Problems:

1. To develop an automated system that extracts entities and relationships from text data using NLP.
2. To construct an interactive knowledge graph that visualizes these connections.
3. To implement semantic search and query answering for intelligent information retrieval.
4. To link related entities across different datasets using similarity-based domain linking.
5. To deploy a user-friendly web application for end-to-end knowledge graph generation and exploration.

2.2 Abstraction:

In today's information-driven era, vast amounts of unstructured data are generated daily in the form of articles, research papers, and online content. Extracting meaningful knowledge from such unstructured textual data remains a significant challenge, as traditional text mining and keyword-based search techniques fail to capture semantic relationships and contextual understanding. To overcome these limitations, this project, titled **"Semantic Knowledge Graph Generator,"** focuses on automatically extracting entities, identifying relationships, and representing them as a structured semantic knowledge graph that can be visually analyzed and semantically queried.

The proposed system utilizes natural language processing (NLP) and machine learning techniques to transform unstructured text into interconnected knowledge representations. The process begins with **data collection and preprocessing**, followed by **Named Entity Recognition (NER)** and **Relation Extraction (RE)** using the SpaCy NLP framework. The extracted entity-relation pairs are then structured into triples and visualized as nodes and edges within a knowledge graph using **NetworkX** and **PyVis** libraries. Furthermore, the project integrates **SentenceTransformer** and **PyTorch** to perform **semantic similarity computation**, enabling cross-domain linking and advanced search capabilities across multiple datasets.

A key component of this system is its **Streamlit-based interactive interface**, which allows users to upload datasets, visualize the generated graphs, and perform semantic search or question-answering operations. By enabling users to query the graph semantically, the system provides an intelligent way to explore and discover relationships among concepts that may not be explicitly connected in the raw data. This makes it highly applicable in domains such as research analysis, education, knowledge management, and data integration.

Overall, the **Semantic Knowledge Graph Generator** serves as a comprehensive framework for knowledge extraction, visualization, and semantic reasoning. It demonstrates how integrating NLP, semantic embedding, and interactive visualization tools can enhance information retrieval and decision-making processes. The system provides a foundation for future advancements in intelligent information systems, offering a scalable and extensible platform for cross-domain knowledge discovery and integration.

CHAPTER III

SYSTEM STUDY

3.1 Existing System:

Existing systems for text analysis and knowledge extraction primarily rely on keyword-based searches or manual tagging. These systems lack contextual understanding and fail to represent complex relationships between entities. Some systems can perform basic named entity recognition but cannot automatically infer semantic relations or link entities across multiple domains. Additionally, most existing visualization tools are static and do not allow real-time interaction or query-based exploration.

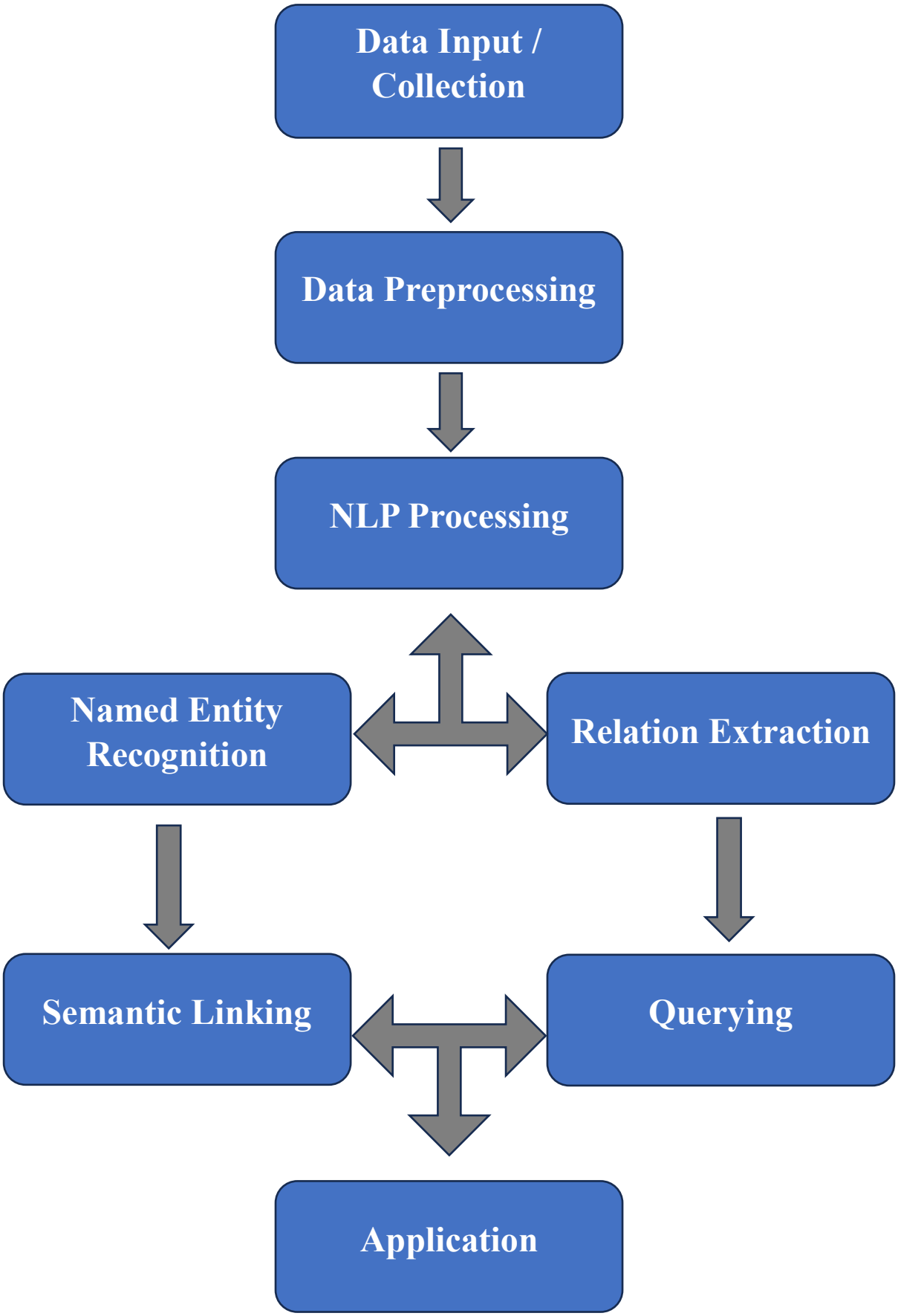
3.2 Proposed System:

The proposed Semantic Knowledge Graph Generator addresses the limitations of existing systems by integrating NLP, deep learning, and graph visualization techniques into a single pipeline. It performs automated entity and relation extraction, builds a knowledge graph, computes semantic similarities, and supports user queries.

3.3 System Architecture:

The system architecture of the Semantic Knowledge Graph Generator consists of multiple stages that work together to process, analyze, and visualize the textual data.

3.3.1 Architecture Diagram:



CHAPTER IV

LIBRARIES USED

4.1 List of Libraries used in the Project:

1. Streamlit
2. Pandas
3. SpaCy
4. NetworkX
5. PyVis
6. SentenceTransformer
7. Torch
8. Tempfile

4.2 About the Libraries:

4.2.1 Streamlit:

Streamlit is an open-source Python library used to create interactive and visually appealing web applications for machine learning and data science projects with minimal coding effort. In this project, Streamlit is used as the **frontend framework** to design the user interface, enabling users to upload datasets, visualize knowledge graphs, and perform semantic searches and queries in real time. It allows seamless integration of Python scripts into a web-based environment, making the system highly interactive and user-friendly.

4.2.2 Pandas:

Pandas is a powerful data manipulation and analysis library in Python. It provides data structures such as **DataFrame** and **Series** for efficient handling of structured data. In this project, Pandas is used for **data collection, preprocessing, and storage** of extracted entities and relations in tabular form. It simplifies operations like reading CSV/Excel files, filtering sentences, and managing triples (Entity1–Relation–Entity2) before visualization and analysis.

4.2.3 SpaCy:

SpaCy is a leading **Natural Language Processing (NLP)** library that offers advanced linguistic analysis tools such as tokenization, part-of-speech tagging, dependency parsing, and named entity recognition (NER). In this project, SpaCy plays a key role in **extracting entities and identifying relationships** between them. It processes unstructured text and transforms it into structured components, enabling relation extraction and knowledge representation.

4.2.4 NetworkX:

NetworkX is a Python library designed for the **creation, manipulation, and study of complex networks and graphs**. It provides a wide range of functions for graph analytics, including centrality measures, shortest paths, and community detection. In the Semantic Knowledge Graph Generator, NetworkX is used to construct and analyze the **knowledge graph** — representing entities as nodes and relationships as edges — and to compute metrics such as **degree centrality** to find key nodes in the network.

4.2.5 PyVis:

PyVis is a visualization library built on top of the **Vis.js JavaScript framework**, used for interactive network visualization in Python. It integrates smoothly with NetworkX and is used here to **visually render the knowledge graph** in a web browser. PyVis enables users to interact with graph nodes, zoom in/out, and explore entity connections dynamically, enhancing the interpretability of semantic relationships.

4.2.6 SentenceTransformer:

SentenceTransformer is a Python library that allows the use of **pre-trained transformer-based models** to generate high-quality sentence and phrase embeddings. In this project, it is used to **calculate semantic similarity** between entities or sentences, enabling cross-domain linking and semantic search. The model "all-MiniLM-L6-v2" is used for encoding textual information into vector form, which helps in measuring meaning-based similarity between textual elements.

4.2.7 Torch (PyTorch):

Torch, or **PyTorch**, is an open-source machine learning framework primarily used for tensor computation and deep learning model deployment. In this project, it is used indirectly through SentenceTransformer for **semantic embedding computations** and **cosine similarity calculations**. Torch ensures efficient numerical computation, allowing the project to process large amounts of text data smoothly, even on CPUs.

4.2.8 Tempfile:

Tempfile is a built-in Python library used to create **temporary files and directories** during program execution. In this project, Tempfile is used to **temporarily store and render the HTML file** generated by the PyVis knowledge graph visualization. This approach improves performance and prevents unnecessary file storage on the system, maintaining efficient and secure data handling during execution.

CHAPTER V

MODULES DESCRIPTION

5.1 List of Modules used in the Project:

1. Data Collection
2. Data Preprocessing
3. Named Entity Recognition (NER)
4. Relation Extraction
5. Knowledge Graph Construction
6. Semantic Search
7. Question Answering
8. Cross-Domain Linking
9. Visualization
10. Deployment

5.2 Explanation of Modules:

5.2.1 Data Collection:

The first step in the system involves collecting textual data from multiple sources such as research articles, documents, or CSV/Excel files. In this project, users can upload datasets containing sentences using the Streamlit interface. The uploaded file is read using the **Pandas** library, which supports both .csv and .xlsx formats. The column labeled “sentence” serves as the primary input for further natural language processing tasks. This flexible data input method allows integration with diverse domains and ensures scalability for large datasets.

5.2.2 Data Preprocessing:

Once the data is collected, preprocessing ensures that it is clean and consistent for accurate analysis. The preprocessing involves text normalization, such as converting text to lowercase, removing unwanted spaces, and eliminating redundant words like "the." This normalization process, implemented through a custom function in the code, helps in reducing noise and ensuring that similar entities (e.g., "France" and "the France") are treated uniformly. The preprocessing step is crucial for improving the accuracy of both entity recognition and relation extraction.

5.2.3 Named Entity Recognition (NER):

Named Entity Recognition is a core module where the system identifies and classifies entities from textual data. Using the **SpaCy** library, the project loads the pre-trained model en_core_web_sm, which automatically detects entities such as people, organizations, locations, and dates. The NER module extracts entities and their types (like GPE, ORG, PERSON) and returns them as structured pairs. These extracted entities form the fundamental nodes of the knowledge graph and provide semantic meaning to the unstructured text.

5.2.4 Relation Extraction:

After identifying entities, the next step is to determine the relationships between them. Relation Extraction (RE) is implemented using SpaCy's dependency parsing feature. The system identifies the **subject**, **verb**, and **object** patterns in each sentence. For instance, in the sentence *"Paris is the capital of France,"* the model extracts the relation (Paris, capital, France). These extracted triples—**Entity1**, **Relation**, and **Entity2**—form the structured representation of knowledge and are stored for later graph visualization and semantic querying.

5.2.5 Knowledge Graph Construction:

This module transforms the extracted triples into a **graph-based representation** using **NetworkX** and **PyVis** libraries. Each entity becomes a node, and each relationship becomes a directed edge connecting the entities. The graph not only visualizes relationships but also allows the computation of network metrics like **degree centrality** and **community detection**. The PyVis library is used to generate an interactive, web-based graph visualization within Streamlit, allowing users to explore entities and their semantic relationships visually.

5.2.6 Semantic Search:

The **Semantic Search** module enables users to query the knowledge graph intelligently. Instead of performing a simple keyword search, it uses **SentenceTransformer** and **PyTorch** to convert both the query and the entities into vector embeddings. The system then computes the **cosine similarity** between these embeddings to find semantically similar concepts. This allows users to retrieve information that matches the meaning of their query, even if the exact words differ. For example, searching “capital city of France” would still retrieve “Paris.”

5.2.7 Question Answering:

In this module, the system allows users to input natural language questions, such as “What is the capital of France?” The question is processed using SpaCy to extract relevant entities and key terms. The system then scans the extracted triples to find matches between the query entities and existing knowledge graph entries. If relevant matches are found, the system returns potential answers derived from the graph. This feature adds an intelligent layer of reasoning and enhances user interactivity by providing direct answers from the knowledge base.

5.2.8 Cross-Domain Linking:

The **Cross-Domain Linking** module is designed to connect related information across different datasets or domains. Using **SentenceTransformer**, the system computes semantic similarity between entity-relation triples and identifies those with high similarity scores. If two sentences or triples from different sources express semantically similar information, they are linked together. This helps in creating an integrated, cross-domain knowledge network that supports comprehensive information discovery and eliminates data silos.

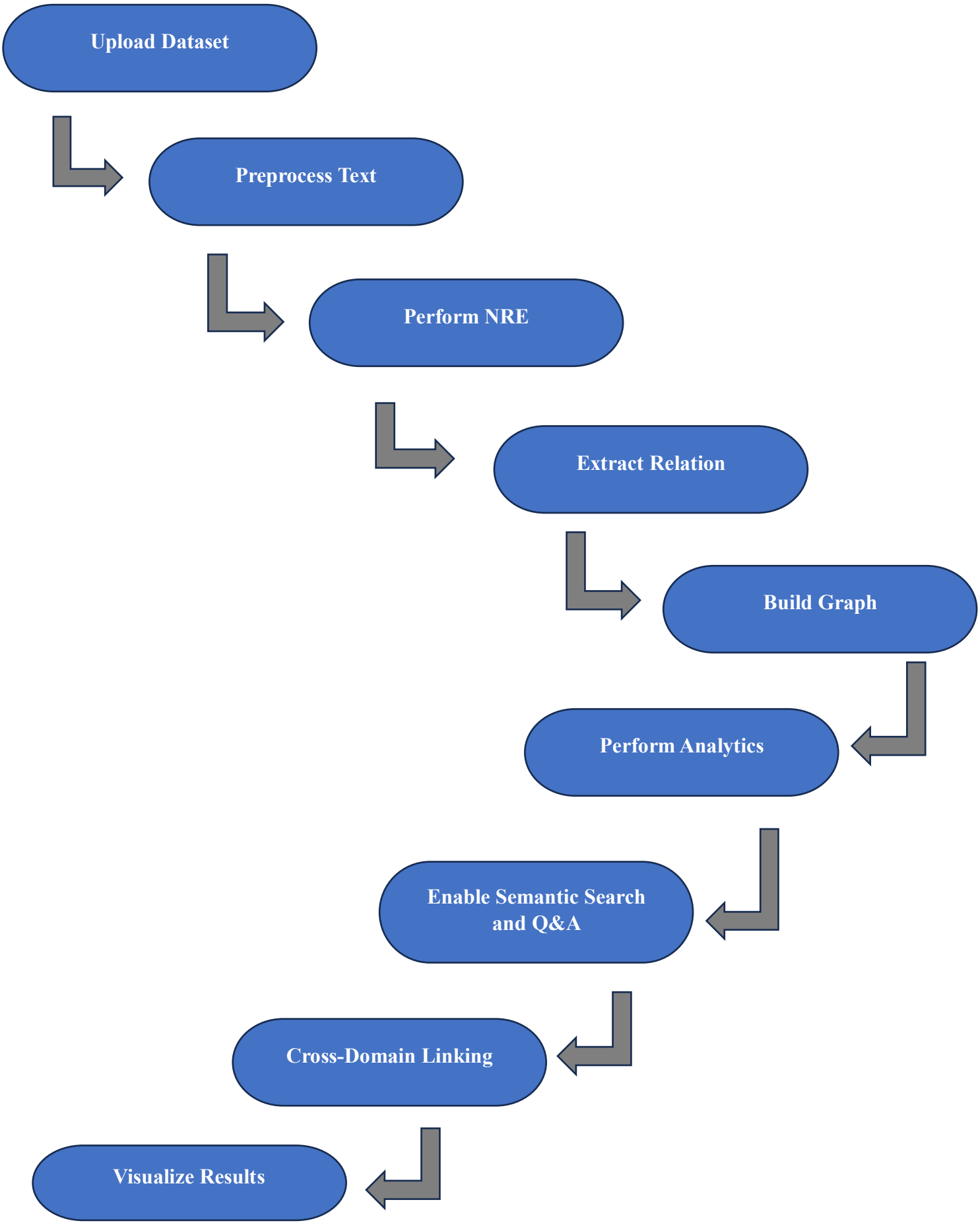
5.2.9 Visualization:

Visualization is a vital part of the project, providing users with an interactive interface to explore the generated knowledge graph. Using PyVis, the system dynamically displays entities as nodes and their relationships as edges, enabling intuitive understanding of data connections. The visualization also highlights important nodes based on centrality and community structure, making it easier to identify key entities or clusters within the data. Users can zoom, drag, and interact with nodes to analyze relationships in real-time through the Streamlit interface.

5.2.10 Deployment:

The final stage of the project focuses on making the system accessible to end users. The project is deployed as a Streamlit web application, which provides a simple and interactive front-end interface. This allows users to upload data, extract entities and relations, visualize the graph, perform semantic searches, and query information — all through a web browser without needing programming expertise. Streamlit's caching and efficient model handling ensure smooth performance even on CPU environments, making the system portable and deployment-ready for educational and research use.

5.3 Methodology and Workflow:



CHAPTER VI
PROJECT CODE

```
import streamlit as st
import pandas as pd
import spacy
import networkx as nx
from pyvis.network import Network
import streamlit.components.v1 as components
import tempfile
from sentence_transformers import SentenceTransformer, util
import torch

# ----------------------------
# Model Loading (Safe CPU mode)
# ----------------------------
@st.cache_resource
def load_models():
    nlp = spacy.load("en_core_web_sm")
    model = SentenceTransformer("all-MiniLM-L6-v2", device="cpu")
    return nlp, model

nlp, model = load_models()

# ----------------------------
# Utility: Normalize Entities
# ----------------------------
```

```
def normalize(text):
    return text.lower().replace("the ", "").strip()


# ----------------------------
# Step 1: Named Entity Recognition
# ----------------------------
def extract_entities(text):
    doc = nlp(text)
    return [(ent.text, ent.label_) for ent in doc.ents]


# ----------------------------
# Step 2: Relation Extraction
# ----------------------------
def extract_relations(text):
    doc = nlp(text)
    relations = []
    for token in doc:
        if token.dep_ == "ROOT" and token.pos_ == "VERB":
            subject = [w.text for w in token.lefts if w.dep_ in ["nsubj", "nsubjpass"]]
            obj = [w.text for w in token.rights if w.dep_ in ["dobj", "attr", "dative",
"oprd"]]
            if subject and obj:
                relations.append((subject[0], token.text, obj[0]))
    return relations


# ----------------------------
# Step 3: Domain Linking (Semantic Similarity)
# ----------------------------
```

```
def link_domains(triples_df, threshold=0.6):
    # Create full semantic phrases from triples
    triples_df["sentence"] = triples_df.apply(
        lambda row: f"{row['Entity1']} {row['Relation']} {row['Entity2']}", axis=1
    )
    sentences = triples_df["sentence"].drop_duplicates().tolist()

    if len(sentences) < 2:
        return []

    embeddings = model.encode(sentences, convert_to_tensor=True)
    linked = []

    for i in range(len(sentences)):
        for j in range(i + 1, len(sentences)):
            score = util.pytorch_cos_sim(embeddings[i], embeddings[j]).item()
            linked.append((sentences[i], sentences[j], round(score, 3)))

    # Show top 10 links
    top_links = sorted(linked, key=lambda x: x[2], reverse=True)[:10]
    return [link for link in top_links if link[2] > threshold]

# ----------------------------
# Step 4: Visualization & Analytics
# ----------------------------
def visualize_knowledge_graph(triples_df, highlight_nodes=None):
    G = nx.DiGraph()
```

```
    for _, row in triples_df.iterrows():
        G.add_node(row["Entity1"])
        G.add_node(row["Entity2"])
        G.add_edge(row["Entity1"], row["Entity2"], label=row["Relation"])

    st.write("### Graph Analytics")
    if len(G.nodes) > 0:
        degree_centrality = nx.degree_centrality(G)
        top_central_nodes = sorted(degree_centrality.items(), key=lambda x: x[1],
reverse=True)[:5]
        st.write("#### Top 5 Central Nodes (by Degree Centrality)")
        for node, score in top_central_nodes:
            st.write(f"- **{node}** → {score:.3f}")

        try:
            from networkx.algorithms.community import
greedy_modularity_communities
            communities = list(greedy_modularity_communities(G))
            st.write(f"#### Detected {len(communities)} Communities:")
            for i, community in enumerate(communities):
                st.write(f"**Community {i+1}:** {list(community)}")
        except Exception as e:
            st.warning(f"Community detection skipped: {e}")
    else:
        st.warning("No nodes available for centrality or community analysis.")

    net = Network(height="600px", width="100%", directed=True)
    for node in G.nodes:
```

```
        net.add_node(node, label=node, color="red" if highlight_nodes and node
in highlight_nodes else None)
    for edge in G.edges(data=True):
        net.add_edge(edge[0], edge[1], label=edge[2]["label"])
    for edge in net.edges:
        edge["title"] = edge["label"]


    with tempfile.NamedTemporaryFile(delete=False, suffix=".html") as
tmp_file:
        net.write_html(tmp_file.name)
        components.html(open(tmp_file.name, "r").read(), height=600)


# ----------------------------
# Streamlit App
# ----------------------------
st.title("Semantic Knowledge Graph Generator (Enhanced Version)")


uploaded_file = st.file_uploader("Upload a CSV or Excel file", type=["csv",
"xlsx"])
triples = []


if uploaded_file:
    df = pd.read_csv(uploaded_file) if uploaded_file.name.endswith(".csv") else
pd.read_excel(uploaded_file)


    if "sentence" not in df.columns:
        st.error("File must contain a column named 'sentence'")
    else:
        st.info("Using column: **sentence**")
```

```
    for text in df["sentence"].dropna():
        triples.extend(extract_relations(text))

    triples_df = pd.DataFrame(triples, columns=["Entity1", "Relation", "Entity2"])
    triples_df.to_csv("triples_output.csv", index=False)
    st.success("Triples extracted and saved to triples_output.csv")

    st.write("### Extracted Triples")
    st.dataframe(triples_df, use_container_width=True, height=400)

    # ----------------------------
    # Semantic Search
    # ----------------------------
    st.write("### Search the Knowledge Graph")
    query = st.text_input("Enter your search query:", "")
    highlight_nodes = None

    if st.button("Search") and query:
        all_nodes = list(set(triples_df["Entity1"].tolist() + triples_df["Entity2"].tolist()))
        node_embeddings = model.encode(all_nodes, convert_to_tensor=True)
        query_embedding = model.encode(query, convert_to_tensor=True)
        cosine_scores = util.pytorch_cos_sim(query_embedding, node_embeddings)[0]
        results = sorted(zip(all_nodes, cosine_scores), key=lambda x: x[1], reverse=True)[:5]

        st.write("### Top Matches:")
```

```
        for node, score in results:
            st.write(f"- {node} ({score:.4f})")
        highlight_nodes = [r[0] for r in results]

    # ----------------------------
    # Query Answering
    # ----------------------------
    st.write("### Query Answering")
    question = st.text_input("Ask a question (e.g., 'What is the capital of France?')")

    if st.button("Get Answer") and question:
        q_doc = nlp(question)
        q_ents = [ent.text for ent in q_doc.ents]
        q_tokens = [token.text for token in q_doc if token.pos_ in ["NOUN", "PROPN"]]

        found = []
        for _, row in triples_df.iterrows():
            if any(ent.lower() in row["Entity1"].lower() or ent.lower() in row["Entity2"].lower()
                   for ent in q_ents + q_tokens):
                found.append(row)

        if found:
            st.write("### Possible Answers:")
            seen = set()
            for _, row in pd.DataFrame(found).iterrows():
```

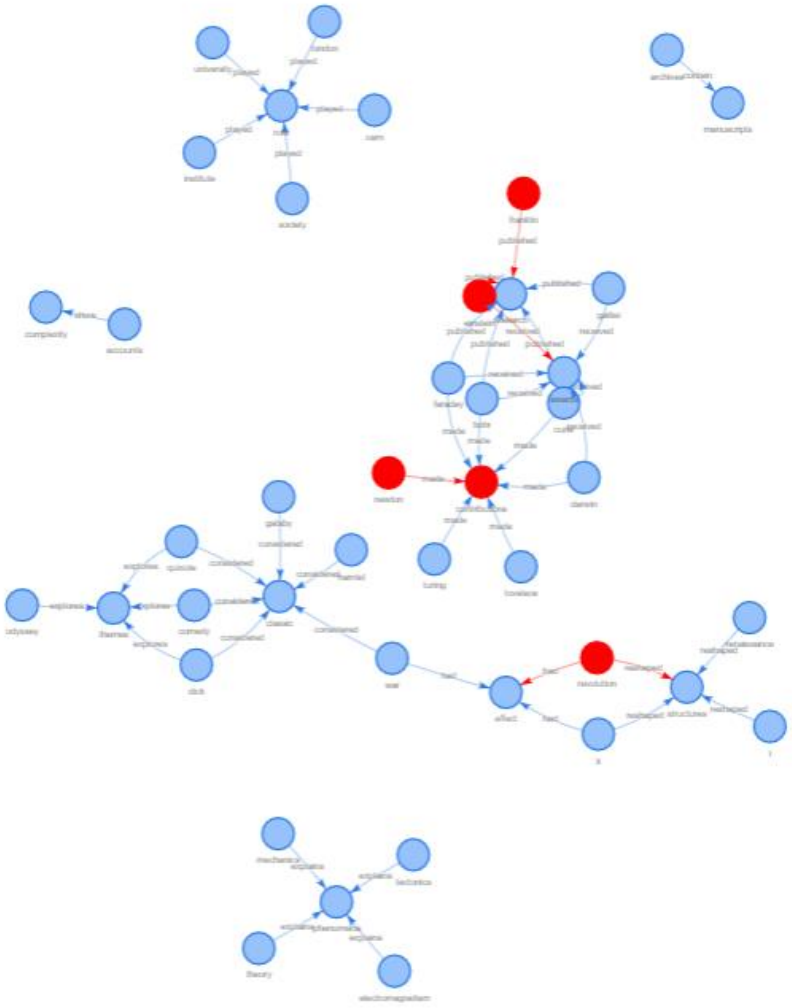
```
            sentence = f"{row['Entity1']} {row['Relation']} {row['Entity2']}"
            if sentence not in seen:
                st.write(f"- **{sentence}**")
                seen.add(sentence)
        else:
            st.write("No direct match found. Try rephrasing your question.")

    # ----------------------------
    # Domain Linking
    # ----------------------------
    st.write("### Domain Linking (Similar Entities)")
    domain_links = link_domains(triples_df)
    if domain_links:
        st.dataframe(pd.DataFrame(domain_links, columns=["Entity1",
"Entity2", "Similarity"]))
    else:
        st.write("No strong semantic links found.")

    # ----------------------------
    # Visualize Graph
    # ----------------------------
    st.write("### Semantic Knowledge Graph Visualization")
    visualize_knowledge_graph(triples_df, highlight_nodes=highlight_nodes)
```

CHAPTER VII

CONCLUSION

7.1 Result and Discussion:

The generated knowledge graphs successfully represent the semantic relationships between entities. Central nodes and communities identified through degree centrality analysis reveal key entities and clusters of related knowledge.



7.2 Future Enhancement:

Future work can focus on integrating larger language models for improved entity recognition and relation extraction. The system can also include domain-specific customization, multilingual support, and integration with databases for persistent knowledge storage. Additionally, incorporating graph-based machine learning algorithms could enhance predictive insights and reasoning capabilities.

7.3 Conclusion:

The project successfully demonstrates the extraction and visualization of semantic relationships from textual data. By combining NLP, machine learning, and interactive visualization, it provides a user-friendly system that converts unstructured data into meaningful insights. The integration of semantic search and question answering enhances knowledge retrieval, while cross-domain linking broadens the understanding of related concepts. Deployment through Streamlit ensures accessibility and real-time interaction.