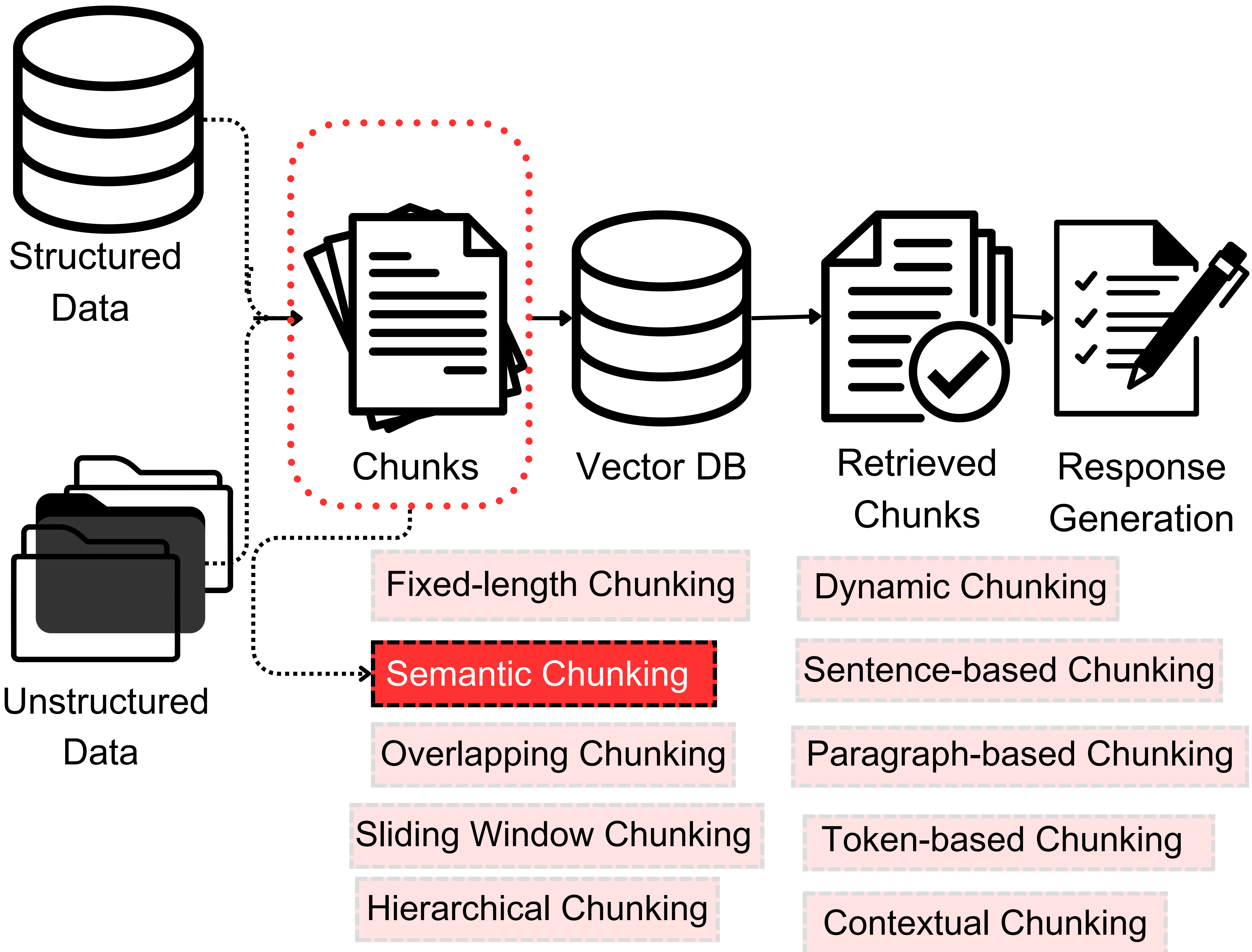
Different Types of

Chunking

in RAG System



Semantic Chunking

Why did Semantic Chunking came into existence?

Preserving Meaning

- Fixed-length chunking often cuts through the middle of sentences or ideas, making it harder for models to understand and process the text correctly.
- By chunking text based on its meaning, each chunk retains a full idea or concept.
- The semantic splitter adaptively picks the breakpoint in-between sentences using embedding similarity.

Contextual Understanding

- In retrieval-augmented tasks, it’s crucial to provide chunks that represent coherent ideas for better response generation.

Semantic Chunking in Python

Steps

1. Install Required Libraries:
   - Make sure you have LlamaIndex installed:

```
%pip install llama-index-embeddings-openai
```

2. Load Your Document:
   - Load your documents into LlamaIndex. This could be a text file, a list of strings, or another supported format.

```
from llama_index.core import SimpleDirectoryReader

# load documents
documents = SimpleDirectoryReader(input_files=["paul_graham_essay.txt"]).load_data()
```
✓ 0.0s

3. Set Up the Semantic Chunker:
   - LlamaIndex provides built-in support for semantic chunking. Use the SemanticTextSplitter to segment your document.

```
from llama_index.core.node_parser import (
    SentenceSplitter,
    SemanticSplitterNodeParser,
)
from llama_index.embeddings.openai import OpenAIEmbedding

import os

os.environ["OPENAI_API_KEY"] = ""
```
✓ 0.0s

Semantic Chunking in Python

4. Setup Embedding & Node Parser with the Semantic Splitter:
    - Use the node parser to split the document into semantically coherent chunks (nodes).

OpenAI Embedding Model

```python
embed_model = OpenAIEmbedding()
splitter = SemanticSplitterNodeParser(
    buffer_size=1, breakpoint_percentile_threshold=95, embed_model=embed_model
)

# also baseline splitter
base_splitter = SentenceSplitter(chunk_size=512)
```

✓ 0.3s

```python
nodes = splitter.get_nodes_from_documents(documents)
```

✓ 17.4s

Semantic Chunking in Python

5. Inspect the Chunks

Chunk 1

IBM related Chunk

Chunk 1: IBM 1401

```python
print(nodes[1].get_content())
```

✓ 0.0s

```
I wrote what beginning writers were supposed to write then, and probably still are: short stories. My stories were awful. They had hardly any plot, just characters with

The first programs I tried writing were on the IBM 1401 that our school district used for what was then called "data processing." This was in 9th grade, so I was 13 or 1

The language we used was an early version of Fortran. You had to type programs on punch cards, then stack them in the card reader and press a button to load the program

I was puzzled by the 1401. I couldn't figure out what to do with it.
```

Chunk 2

Personal Computer + College

Chunk 2: Personal Computer + College

```python
print(nodes[2].get_content())
```

✓ 0.0s

```
And in retrospect there's not much I could have done with it. The only form of input to programs was data stored on punched cards, and I didn't have any data stored on pun

With microcomputers, everything changed. Now you could have a computer sitting right in front of you, on a desk, that could respond to your keystrokes as it was running in

The first of my friends to get a microcomputer built it himself. It was sold as a kit by Heathkit. I remember vividly how impressed and envious I felt watching him sitting

Computers were expensive in those days and it took me years of nagging before I convinced my father to buy one, a TRS-80, in about 1980. The gold standard then was the App

Though I liked programming, I didn't plan to study it in college. In college I was going to study philosophy, which sounded much more powerful. It seemed, to my naive high

I couldn't have put this into words when I was 18. All I knew at the time was that I kept taking philosophy courses and they kept being boring. So I decided to switch to AI

AI was in the air in the mid 1980s, but there were two things especially that made me want to work on it: a novel by Heinlein called The Moon is a Harsh Mistress, which fe

There weren't any classes in AI at Cornell then, not even graduate classes, so I started trying to teach myself. Which meant learning Lisp, since in those days Lisp was re
```

Semantic Chunking in Python

Chunk 3

Finishing up College + Grad School

Chunk 3: Finishing up College + Grad School

```python
print(nodes[3].get_content())
```

✓ 0.0s Python

```
I knew what I was going to do.

For my undergraduate thesis, I reverse-engineered SHRDLU. My God did I love working on that program. It was a pleasing bit of code, but what made it even more exciting was

I had gotten into a program at Cornell that didn't make you choose a major. You could take whatever classes you liked, and choose whatever you liked to put on your degree.

I applied to 3 grad schools: MIT and Yale, which were renowned for AI at the time, and Harvard, which I'd visited because Rich Draves went there, and was also home to Bill

I don't remember the moment it happened, or if there even was a specific moment, but during the first year of grad school I realized that AI, as practiced at the time, was
```

- Overcomes Token Limits:
  - Semantic chunking ensures long documents fit within these constraints.
- Improves Retrieval Accuracy:
  - Smaller, well-defined chunks increase the precision of information retrieval systems.
- Context Awareness:
  - Chunking ensures the system doesn't lose important context, enabling more accurate responses.
- Query Relevance:
  - Helps match user queries to relevant sections of text by indexing smaller, focused chunks.

Semantic Chunking in Python

Using LangChain - Complete Code

```python
from langchain.embeddings import HuggingFaceEmbeddings
from langchain.vectorstores import FAISS
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain.chat_models import ChatOpenAI
from langchain.document_loaders import TextLoader
from langchain.chains import RetrievalQA
from langchain.llms import OpenAI  # Import LLM class
import os

# Set your OpenAI API key
os.environ["OPENAI_API_KEY"] = ""  # Replace with your API key

# Initialize the LLM
llm = OpenAI(model_name="gpt-3.5-turbo", temperature=0)  # Choose your model and parameters

!wget 'https://raw.githubusercontent.com/run-llama/llama_index/main/docs/docs/examples/data/paul_graham/paul_graham_essay.txt' -O 'pg_essay.txt'
```

```python
# Load documents (Replace with your own data loading logic)
def load_documents():
    loader = TextLoader("pg_essay.txt")  # Path to your text file
    documents = loader.load()
    return documents

# Perform semantic chunking
def semantic_chunking(documents, chunk_size=512, chunk_overlap=50):
    text_splitter = RecursiveCharacterTextSplitter(
        chunk_size=chunk_size,
        chunk_overlap=chunk_overlap
    )
    chunks = []
    for doc in documents:
        chunks.extend(text_splitter.split_text(doc.page_content))
    return chunks

# Embed and store the chunks
def embed_and_store(chunks):
    embeddings = HuggingFaceEmbeddings(model_name="sentence-transformers/all-MiniLM-L6-v2")
    vector_store = FAISS.from_texts(chunks, embeddings)
    return vector_store
```

Semantic Chunking - LangChain

```python
# Main RAG pipeline
def rag_pipeline():
    # Load and chunk documents
    documents = load_documents()
    chunks = semantic_chunking(documents)

    # Embed and store
    vector_store = embed_and_store(chunks)

    # Set up retriever
    retriever = vector_store.as_retriever()

    # Initialize LLM
    llm = ChatOpenAI(
        model="gpt-3.5-turbo",
        temperature=0,
        max_tokens=256
    )

    # Set up a QA chain
    qa_chain = RetrievalQA.from_chain_type(
        llm=llm,  # Provide the language model
        retriever=retriever,
        chain_type="stuff",
        return_source_documents=True
    )

    return qa_chain
```

Semantic Chunking - LangChain

```python
if __name__ == "__main__":
    qa_chain = rag_pipeline()
    query = "What is semantic chunking?"

    # Use __call__ instead of run to get all outputs
    result = qa_chain({"query": query})  # Pass the query as a dictionary

    # Extract the results
    answer = result["result"]
    source_documents = result["source_documents"]

    # Print the outputs
    print("Answer:", answer)
    print("\nSource Documents:")
    for doc in source_documents:
        print(doc.page_content)
```

https://github.com/DataSphereX/Chunking-Strategies

DataSphereX/**Chunking-Strategies**

1 Contributor | 0 Issues | 0 Stars | 0 Forks

DataSphereX/Chunking-Strategies

Contribute to DataSphereX/Chunking-Strategies development by creating an account on GitHub.

GitHub

CONGRATULATIONS

You have reached the end, now

If you want to help your network

REPOST THIS

Sarveshwaran R