LlamaIndex Document Chat System Workflow

Overview

LlamaIndex is an alternative to LangChain that helps you build applications using Large Language Models (LLMs). The main use case shown here is creating a system where you can ask questions about your documents and get AI-powered answers.

System Architecture - The 4 Main Steps

1. Document Loading

- Load your documents (text files, PDFs, Word docs, etc.)
- Documents are stored in a folder structure
- LlamaIndex automatically detects file types and uses appropriate loaders

2. Document Chunking & Embedding

- Documents are divided into smaller chunks (pieces of text)
- Each chunk gets converted into embeddings (numerical representations)
- Embeddings capture the semantic meaning of the text (what meaning do the words describe)
- These embeddings are stored in a vector database.

3. Query Processing

- When you ask a question, the system converts your question into embeddings
- It performs a semantic search to find chunks most relevant to your question
- By default, it returns to the top 2 most relevant chunks

4. Answer Generation

- The relevant chunks serve as context for the LLM
- The LLM receives both your question AND the relevant context
- It generates a response based on this information
-

Implementation - The 4 Lines of Code

1. Load documents

```
documents = SimpleDirectoryReader("data").load_data()
```

2. Create vector store index (handles chunking + embeddings)
index = VectorStoreIndex.from_documents(documents)
3. Create query engine
query_engine = index.as_query_engine()
4. Ask questions
response = query_engine.query("What did the author do growing up?")

Index:

The **index** is essentially a processed, searchable version of your documents. Containing the following below

1. Your Original Text Chunks

Chunk 1: "The author worked on writing and programming outside of school..."
Chunk 2: "Before college, he spent time creating short stories..."
Chunk 3: "He tried programming on an IBM 1401 computer..."

2. Embeddings (Numerical Representations)

Chunk 1 → [0.2, 0.8, 0.1, 0.9, -0.3, 0.7, ...] (1536 numbers)
Chunk 2 → [0.1, 0.6, 0.3, 0.8, -0.1, 0.5, ...] (1536 numbers)
Chunk 3 → [0.4, 0.2, 0.7, 0.1, -0.8, 0.9, ...] (1536 numbers)

3. Mapping Information

- Which embedding belongs to which chunk
- Metadata about each chunk (source document, position, etc.)

Key Features & Customizations

Persistence

- Save your index to disk using index.persist()
- Load it later using load_index_from_storage()
- Avoids re-processing documents every time

Adjust Chunk Size:

Set custom chunk size (How many tokens (roughly-words) you split each document into before creating embeddings.) and overlap.

Note:

a. Too Small Chunks (e.g.: 50 tokens) may cause:
    1. Important information gets fragmented
    2. LLM gets pieces but misses the full picture
    3. More chunks = more embeddings to store
b. Too Large Chunks (e.g.: 2000 tokens) may cause:
    1. Chunk contains relevant info + lots of irrelevant stuff
    2. Embedding represents mixed topics, harder to find
    3. Fewer chunks fit in LLM's context window

Here is the code to change chunk size and overlap:

```
service_context = ServiceContext.from_defaults(
  chunk_size=1000,
  chunk_overlap=20
)
```