Scheme Research Tool - Technical Implementation Report

By Bhavya Bhargava (3/01/2025)

Documents for Submission

- **Solution Files -** https://drive.google.com/drive/folders/1u0AlWnIlLK2lz4XnUBcEQPtDalekTkRs?usp=drive_link
- **Video Demonstration (unlisted video)–** https://youtu.be/Dd-Q0enb6OQ

Project Overview

The Scheme Research Tool is a Streamlit-based web application designed to automate the process of analyzing government scheme information. The tool takes URLs of scheme articles as input, processes them to create structured summaries, and enables users to ask questions about the schemes through an interactive interface.

Technical Architecture

1. Core Technologies Used

1.1 Language Model Selection

- **Selected Model**: Groq's LLaMA 3 (8B parameters)
- **Justification**:
    - High performance for text understanding and generation
    - Lower latency compared to other models
    - Cost-effective for production deployment
    - Robust handling of context-based questions
    - Strong performance in summarization tasks

1.2 Embedding Model

- **Selected Model**: Sentence Transformers (all-mpnet-base-v2)
- **Justification**:
    - More cost-effective than OpenAI embeddings as it runs locally without API costs
    - Comparable performance to OpenAI embeddings for semantic similarity tasks
    - No dependency on external API availability or rate limits
    - Complete control over the embedding process

- Faster processing due to elimination of API latency
- Future-proof solution as it's not tied to any specific vendor
- Well-documented and actively maintained by the HuggingFace community
- Supports offline operation for environments with limited connectivity

1.3 Vector Store

- **Selected Solution**: FAISS (Facebook AI Similarity Search)
- **Justification**:
  - Highly efficient similarity search capabilities
  - Excellent scalability for large document collections
  - Low memory footprint
  - Fast retrieval times
  - Robust integration with LangChain

2. Implementation Details

2.1 Document Processing Pipeline

```python
def process_urls(urls):
    try:
        loader = UnstructuredURLLoader(urls=urls)
        documents = loader.load()
        text_splitter = RecursiveCharacterTextSplitter(
            chunk_size=1000,
            chunk_overlap=200,
            length_function=len
        )
        texts = text_splitter.split_documents(documents)
        return texts
    except Exception as e:
        st.error(f"Error processing URLs: {str(e)}")
        return None
```

- Uses UnstructuredURLLoader for robust content extraction
- Implements RecursiveCharacterTextSplitter with optimal chunk sizes
- Handles various document formats and structures

- Includes error handling for failed URL processing

2.2 Vector Store Implementation

```python
def create_vector_store(texts):
    embeddings = HuggingFaceEmbeddings(
        model_name="sentence-transformers/all-mpnet-base-v2"
    )
    vector_store = FAISS.from_documents(texts, embeddings)

    with open("faiss_store_Meta.pkl", "wb") as f:
        pickle.dump(vector_store, f)
    return vector_store
```

- Creates and maintains a FAISS index for efficient similarity search
- Implements persistent storage through pickle serialization
- Ensures quick retrieval for subsequent queries
- Uses HuggingFace embeddings for cost-effective and reliable performance

2.3 Summary Generation

```python
def generate_scheme_summary(vector_store):
    summary_aspects = {
        "benefits": "What are the main benefits of this scheme?",
        "process": "What is the application process for this scheme?",
        "eligibility": "What are the eligibility criteria for this scheme?",
        "documents": "What documents are required for this scheme?"
    }
```

- Structured approach to summary generation
- Covers all required aspects of scheme information
- Uses RAG (Retrieval Augmented Generation) for accurate summaries
- Maintains consistency across different schemes

3. User Interface Implementation

3.1 Input Handling

- Flexible URL input methods:

    - Direct URL input through text area
    - Bulk URL upload through text files
- Real-time processing status updates
- Error handling and user feedback

3.2 Results Display

- Organized summary presentation in four sections:
    - Benefits
    - Application Process
    - Eligibility
    - Required Documents
- Interactive Q&A interface
- Historical query tracking

4. Solution Acceptance Criteria Fulfillment

Requirement	Implementation	Status
Browser-based web app	Implemented using Streamlit	✓
URL input functionality	Sidebar input area with text and file upload options	✓
Data processing trigger	"Process URLs" button with progress indicators	✓
Text splitting and embedding	RecursiveCharacterTextSplitter with HuggingFace embeddings	✓
FAISS indexing	Implemented with pickle storage	✓
Interactive Q&A	RetrievalQA chain with context-aware responses	✓

5. Technical Dependencies

```
streamlit
langchain
langchain-community
faiss-cpu
unstructured
groq
sentence-transformers
langchain_groq
langchain_huggingface
```

6. Configuration

- API keys stored in .config file
- FAISS index stored in faiss_store_Meta.pkl
- Modular design for easy configuration updates

Conclusion

The implemented Scheme Research Tool successfully meets all specified requirements while providing additional features for enhanced usability. The choice of technologies ensures robust performance, scalability, and maintainability. The combination of Groq's LLM, Sentence Transformers for embeddings, and FAISS for vector storage provides an efficient and effective solution for scheme research automation.

Future Improvements

1. Enhanced error handling and recovery mechanisms
2. Additional summary aspects based on user feedback
3. Performance optimizations for larger datasets
4. Integration with additional data sources

This implementation provides a solid foundation for automating scheme research while maintaining accuracy and user-friendliness.