Kalpit Pvt Ltd - AI Engineering Intern (07/12/2025)

Subject: Congratulations & Technical Assignment (Next Round).

Dear Candidate,
Congratulations on clearing the first round!

Next Step: Technical Assignment

NOTE - From now-onwards, you need to do all the communication through e-mail/whatsapp.

***Technical Assignment*: Build a RAG-Based Q&A System based on SEMRAG research paper (Attached).**

Objective
Build a fully functional RAG system following the SEMRAG research paper's approach (**Attached**), that can answer questions about Dr. B.R. Ambedkar's works using the provided 94-page dataset. You must implement the exact architecture described in the paper and demonstrate it working live during the interview.
You don't have to execute the evaluation part of the SEMRAG paper.

NOTE - *those candidates who will be not be able to run code locally during interview, for whatsoever reasons, will be considered disqualified.*

Provided Dataset - See Attached Files

- **Ambedkar_book.pdf** : 94-page PDF of Dr. B.R. Ambedkar's book

---

Requirements from SemRAG Paper (Attached)

1. **Semantic Chunking**
   - Use **cosine similarity** of sentence embeddings to group sentences into semantically coherent chunks.
   - Implement **buffer merging** to preserve contextual continuity.
   - Ensure chunks respect **token limits** (max 1024 tokens, sub-chunks ~128 tokens with overlap).
2. **Knowledge Graph Construction**
   - Extract **entities** and **relationships** from chunks.
   - Build a **graph structure** with nodes (entities) and edges (relationships).
   - Use **community detection** (e.g., Leiden algorithm) to group related entities.
3. **Retrieval Strategies**
   - Implement **Local RAG Search**: Retrieve relevant entities and chunks.
   - Implement **Global RAG Search**: Retrieve top-K community summaries.
   - Use **similarity thresholds** and **ranking** to filter results.
4. **Integration with LLM**
   - Use an **LLM** (e.g., Llama3, Mistral, Gemma2) for summarization and response generation.
   - Implement **prompt engineering** to integrate retrieved graph info into queries.

Tech Stack Suggestions

- Python 3.9+
- Libraries: langchain, sentence-transformers, spaCy, networkx, community (for Louvain), ollama/llama-cpp-python, pypdf, ragas (optional)

1. Core NLP & Embeddings
   - Sentence Embeddings: sentence-transformers (for Algorithm 1)
     - Model: all-MiniLM-L6-v2 or similar
   - NER & Dependency Parsing: spaCy (en_core_web_sm)
   - Text Processing: nltk, transformers
2. Knowledge Graph & Graph Processing
   - Graph Library: networkx or igraph
   - Community Detection: python-louvain or leidenalg
   - Vector Storage: chromadb, faiss, or pinecone (local)
3. LLM Integration
   - Local LLM: ollama (with llama3:8b or mistral:7b)
   - LLM Wrapper: langchain (optional but recommended)
   - Prompt Engineering: Custom templates
4. Retrieval & Search
   - Similarity Search: scikit-learn for cosine similarity
   - Ranking: Custom re-ranking implementation
   - Hybrid Search: Combine vector + keyword search
5. Infrastructure
   - Python: 3.9+
   - Environment: conda or venv
   - Containerization: Docker (optional but recommended)
   - Version Control: git

Building the System

Tasks

1. Data Preparation & Semantic Chunking

- Load the provided PDF (Ambedkar_book.pdf) (Attached).
- Implement **Algorithm 1** from the SEMRAG paper (semantic chunking via cosine similarity).
- Use a **local embedding model** (e.g., all-MiniLM-L6-v2 from SentenceTransformers).
- Apply **buffer merging** and enforce token limits.

2. Knowledge Graph Construction

- Use an **NER model** (e.g., spaCy) to extract entities from chunks.
- Extract relationships using **dependency parsing** or a relation extraction model.
- Build a **networkx** or **neo4j** graph with:
  - Nodes = entities
  - Edges = relationships
- Implement **community detection** (Leiden or Louvain) to group entities.

3. Retrieval Implementation

- Implement **Local Graph RAG Search** (Equation 4, SEMRAG Paper). (Attached)
- Implement **Global Graph RAG Search** (Equation 5, SEMRAG Paper). (Attached)
- Use **similarity scoring** (cosine similarity) between query embeddings and chunk/community embeddings.

4. LLM Integration & Response Generation

- Use a **local LLM** (e.g., Llama3-8B via Ollama, or Mistral-7B).
- Implement **prompt templates** that integrate retrieved entities, chunks, and community summaries.
- Generate answers to user queries.

You don't have to execute the evaluation part of the SEMRAG paper.

Implementing the System

Knowledge Graph Construction

Build the graph as described in Sections 3.2.2-3.2.3: (See Attached SEMRAG Research Paper).

```
python
def build_knowledge_graph(chunks):
    """
    Steps:
    1. Extract entities from each chunk using spaCy
    2. Extract relationships between entities
    3. Build graph with nodes (entities) and edges (relationships)
    4. Apply community detection (Leiden/Louvain algorithm)
    5. Generate community summaries using LLM
    6. Store graph with embeddings for retrieval
    """
```

Deliverable: Knowledge graph with:

- Nodes = Entities mentioned in Ambedkar's works
- Edges = Relationships between them
- Communities = Thematically grouped entities
- Community summaries = LLM-generated summaries of each community

Retrieval System Implementation

Implement both retrieval methods exactly as in Equations 4 & 5: (See Attached SEMRAG Research Paper).

python

```
def local_graph_rag_search(query, history=None, top_k=5):
    """
    Implement Equation 4:
    D_retrieved = Top_k({v ∈ V, g ∈ G | sim(v, Q+H) > τ_e ∧ sim(g, v) > τ_d})

    Steps:
    1. Calculate similarity between query and entities
    2. Filter by threshold τ_e
    3. Find chunks related to those entities
    4. Filter by threshold τ_d
    5. Return top_k results
    """

def global_graph_rag_search(query, top_k=3):
    """
    Implement Equation 5:
    D_retrieved = Top_k(⋃_{r ∈ R_Top-K(Q)} ⋃_{c_i ∈ C_r} (⋃_{p_j ∈ c_i} (p_j, score(p_j, Q))), score(p_j, Q))

    Steps:
    1. Find top-K community reports relevant to query
    2. Extract chunks from those communities
    3. Score each point within chunks
    4. Return top-K points based on scores
    """
```

4: LLM Integration & Response Generation

python

```
def generate_answer(query, retrieved_local, retrieved_global):
    """
    Steps:
    1. Combine retrieved local entities and global community summaries
    2. Create prompt with context, entities, and query
    3. Use local LLM (Llama3/Mistral) to generate answer
    4. Include citations to source chunks
    """
```

You don't have to execute the evaluation part of the SEMRAG paper.

These are suggestions, and you have flexibility, but the core approach of SEMRAG research paper (Attached) has to be preserved.

NOTE - *those candidates who will be not be able to run code locally during interview, for whatsoever reasons, will be considered disqualified.*

Project Structure

text

```
ambedkargpt/
├── data/
│   ├── Ambedkar_works.pdf
│   └── processed/
│       ├── chunks.json
│       └── knowledge_graph.pkl
├── src/
│   ├── chunking/
│   │   ├── semantic_chunker.py   # Algorithm 1
│   │   └── buffer_merger.py
│   ├── graph/
│   │   ├── entity_extractor.py
│   │   ├── graph_builder.py
│   │   ├── community_detector.py
│   │   └── summarizer.py
│   ├── retrieval/
│   │   ├── local_search.py      # Equation 4
│   │   ├── global_search.py     # Equation 5
│   │   └── ranker.py
│   ├── llm/
│   │   ├── llm_client.py
│   │   ├── prompt_templates.py
│   │   └── answer_generator.py
│   └── pipeline/
│       └── ambedkargpt.py       # Main pipeline
├── tests/
│   ├── test_chunking.py
│   ├── test_retrieval.py
│   └── test_integration.py
├── config.yaml                  # Hyperparameters
├── requirements.txt
├── setup.py
└── README.md
```

Getting Started Instructions

1. Set up environment with required tech stack
2. Implement Algorithm 1 for semantic chunking
3. Build knowledge graph with entities/relationships
4. Implement local and global search (Equations 4 & 5, SEMRAG paper) (Attached)
5. Integrate with local LLM
6. Test with provided Ambedkar_book.pdf (Attached)
7. Prepare for live demo

Note: The candidate must bring their own laptop with the system fully installed and ready to run during the interview.

Deliverables

Minimum Viable Product (Must Have):

1. Working semantic chunking on Ambedkar_book.pdf (Attached)
2. Knowledge graph with entities
3. Local search (Equation 4) working
4. LLM answering simple questions
5. Live demo with 3-5 sample questions

Preferred-

1. Both local and global search working
2. Community detection implemented
3. Clean, modular code with tests
4. Configurable parameters (buffer size, thresholds)
5. Error handling and logging

Timeline: 10 days

Further time can be given to the candidate in case of unexceptional contingency, provided an e-mail is sent providing an exact date by which candidate will send the assignment and be ready for the live demonstration during interview.

Submission Requirements -

1. Private GitHub repository with complete, runnable code
2. README.md with setup and usage instructions
3. requirements.txt for dependencies

4. Demo script that can be run during interview
5. Brief documentation of your approach.

Submission to be done at - kalpiksingh2005@gmail.com
+44 -7306687100

NOTE - From Now-Onwards, you need to do all the communication through e-mail/whatsapp.

Next Steps-

1. Complete and submit the assignment on the e-mail provided.
2. We'll review your code and system design
3. You'll be invited for a final interview
4. During the interview, you'll run your system live and answer technical questions

If you have any questions about the process, feel free to reply to this email.

IMPORTANT - those candidates who will be not be able to run code locally during interview, for whatsoever reasons, will be considered disqualified.

Best regards,
The Hiring Team
07/12/2025

kalpiksingh2005@gmail.com

+44 -7306687100