Inter-IIT Tech Meet 14.0

Problem Statement: Semantic Tree-Based Document Retrieval

1 Objective

Design a hierarchical semantic retrieval system that organizes research papers into a tree structure and performs efficient routing-based retrieval. The challenge evaluates your system's retrieval accuracy and the quality of your semantic tree.

2 Provided Inputs

Participants receive:

- 203 PDF research papers
- **metadata.csv**: Document metadata + 384-d embeddings
- **queries_train.jsonl**: Train queries with ground truth
- **queries_val.jsonl**: Validation queries (no ground truth)
- **flat_retrieval.py**: Baseline flat similarity search
- **evaluate_local.py**: Local scoring (P@3, P@5, MRR)
- **tree_schema.json**: Schema for validating `tree.json`

3 Participant Tasks

3.1 Build a Semantic Tree

You must construct a hierarchical semantic tree:

- Internal nodes store centroid embeddings + metadata
- Leaf nodes represent individual documents
- Depth recommended: 3–5
- Clusters must be built using document embeddings (not filenames or metadata)

3.2 Tree-Based Retrieval

Your retrieval pipeline must:

1. Encode queries using **all-MiniLM-L6-v2**
2. At each internal node, compare query vector to child centroids
3. Select top-k children (recommended $k = 3$–5)
4. Recursively traverse until reaching leaves
5. Return top-k most relevant document IDs

   Produces: `query_results_tree.jsonl`

3.3 Flat Retrieval

Run the provided baseline:

- Compute cosine similarity over all documents
- Output: `query_results_flat.jsonl`

(Not graded—only for reference.)

4 Expected Outputs

4.1 tree.json

```
{
  "name": "root",
  "centroid": [...],
  "size": 203,
  "children": {
      "Cluster_1": {...},
      "Cluster_2": {...}
  }
}
```

Requirements:

- Embeddings must remain 384-dimensional
- File size < 100MB
- Leaf nodes contain: id, file$_n$*ame*, *embedding*
- Internal nodes contain: centroid, size, children

4.2 query__results__tree.jsonl

```
{"query_id": "q101", "results": ["docA", "docB", "docC", "docD", "docE
   "]}
```

Exactly 5 predictions per query.

4.3 run.sh

Runs the full pipeline:

- Builds tree
- Runs tree retrieval
- Runs flat baseline
- Completes within 30 minutes

5 Evaluation Metrics

Judges evaluate using hidden test queries.

5.1 Retrieval Quality (80%)

- **Precision@3** (30%)
- **Precision@5** (20%)
- **MRR** (30%)
- **Routing Accuracy** (20%)

Metrics computed using:

```
python evaluate_judge.py results.jsonl queries_test.jsonl tree.json
```

5.2 Tree Quality (10%)

Manual + automated inspection:

- Semantic coherence
- Cluster balance
- Interpretability

5.3 Efficiency (5%)

Latency and memory usage.

5.4 Reproducibility (5%)

- run.sh must execute successfully
- Outputs must be correctly formatted

6 Technical Constraints

6.1 Allowed

- numpy, pandas, sklearn, scipy
- sentence-transformers (must use all-MiniLM-L6-v2)

6.2 Prohibited

- Vector DBs (FAISS, Pinecone, etc.)
- LangChain, LlamaIndex, RAG frameworks
- Using ground truth in tree building
- LLM-based answer generation

7 Submission Format

```
team_name/
          tree.json
          query_results_tree.jsonl
          query_results_flat.jsonl
          run.sh
          src/
          report.pdf
          README.md
```

Submit as `teamname_submission.zip`

End of Problem Statement
