Kautilya ML Talent Challenge

💼Introduction

Welcome to the **Kautilya Machine Learning Challenge**.

Your objective is simple:

- Solve **2 core ML tasks**.
- Use **any programming language**.
- Python is expected, but C++ / Rust / Go implementations earn respect.
- Highest cumulative score → shortlisted for interviews.

These tasks are chosen to test **reasoning**, **ML engineering ability**, **semantic understanding**, and **practical problem-solving**.

🧰The Two Tasks

✂️Semantic Search on Twitter API Documentation

Points: 30

Goal: Build a semantic search engine over the full Twitter API Postman documentation.**

Invocation Requirement

Your solution **must allow command-line querying**:

```
python semantic_search.py --query "How do I fetch tweets with expansions?"
```

This should print the top-k most relevant documentation chunks.

Implementation Requirements

- Use the GitHub repo: https://github.com/xdevplatform/postman-twitter-api
- Chunk documentation intelligently.
- Embed chunks (model of your choice).
- Build a vector index (FAISS/Chroma/custom).
- Implement top-k semantic retrieval.

Output: JSON printed to stdout with ranked chunks.

✒️Narrative Building from 84MB News Dataset

Points: 70

Goal: Load the 84MB JSON news dataset → filter by **source_rating > 8** → extract all stories relevant to **ANY topic the user passes via command-line** → generate a narrative.

Invocation Requirement

Your script must support **dynamic topic generation**:

```
python narrative_builder.py --topic "Jubilee Hills elections"
python narrative_builder.py --topic "Israel-Iran conflict"
python narrative_builder.py --topic "AI regulation"
```

It should build a narrative for *any* topic.

Expected Output Structure

Same structure regardless of topic:

1. Narrative Summary

5–10 sentences synthesizing the topic storyline.

2. Timeline of Events

Chronological ordering of all relevant articles with:

- date
- headline
- url
- why_it_matters

3. Narrative Clusters

Group semantically similar articles into themes.

4. Narrative Graph

Nodes = articles. Edges = relations (builds_on, contradicts, adds_context, escalates).

Final Output JSON

```
{
  "narrative_summary": "...",
  "timeline": [...],
  "clusters": [...],
```

```
  "graph": {...}
}
```

🕐Performance Metrics

To ensure fair and consistent evaluation, the following performance metrics apply **across all tasks**:

1. Correctness & Functionality — 50%

- Does the system produce correct, relevant, logically consistent results?
- For semantic search: Are the top-k results actually meaningful?
- For narrative extension: Do mappings make semantic sense?
- For deduplication: Are duplicates grouped correctly?

2. Performance & Efficiency — 25%

- Vector search speed
- Embedding pipeline efficiency
- Memory handling
- Reasonable runtime given dataset size
- Bonus respect for C++/Rust/Go optimizations

3. Code Quality — 10%

- Clean structure
- Modular functions
- Readability
- Maintainability

4. Extra Credit (Non-Python Implementations) — 15%

- C++ / Rust / Go / Java implementations get bonus

🕐Scoring Breakdown

Task	Points
Semantic Search	**30**
Narrative Building	**70**
Total	**100**