Problem Statement:

Task Objective

Design and implement two distinct news article recommendation algorithms that leverage a user's past reading behavior. The algorithms should consider:

- User's reading history
- User's expressed interests
- Popularity of news articles
- Relevance of articles to the user's current location

The developed algorithms must be capable of real-time recommendations and suitable for deployment in a production environment.

Data:

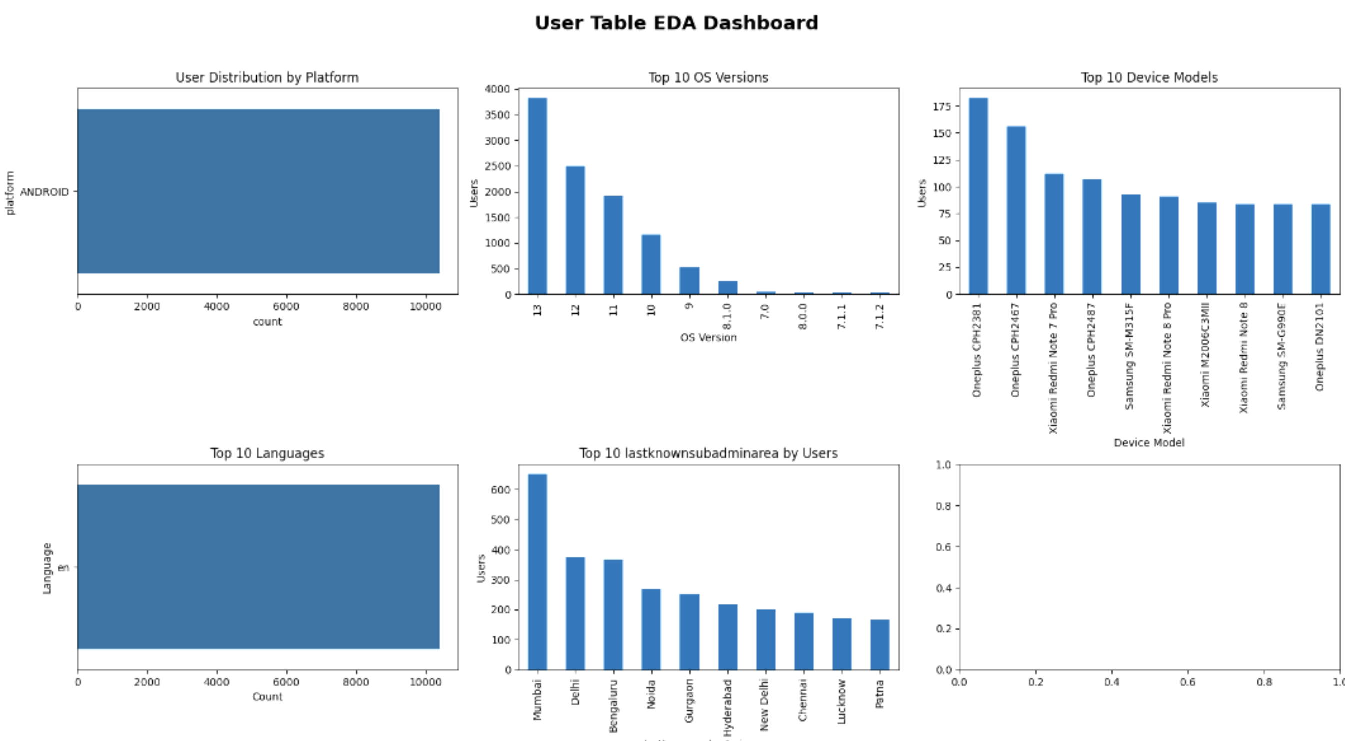
- Device Data : gs://nis-interview-task-data/devices
- Event Data : gs://nis-interview-task-data/event
- Testing Content : gs://nis-interview-task-data/testing_content
- Training Content : gs://nis-interview-task-data/training_content

Exploratory Data Analysis:

- ### User

The dataset contains **10,400 user records** with **11 columns** capturing device, platform, location, language, and activity details.

- **Identifiers & device info**:
  Every user has a unique `deviceid`. All users have Android phones. Device `model` is available for most users (99.5%) but missing for a small fraction (~56 records).
- **Network info**:
  The `networkType` column is almost complete (only 48 missing).
- **Location fields**:
  `district` is **mostly missing** (only 21 non-null values out of 10,400), so it's not reliable. In contrast, `lastknownsubadminarea` (user's city) is much better populated, available for ~91% of users.
- **Language preference**:
  The `language_selected` is not useful as the only language available is `en`

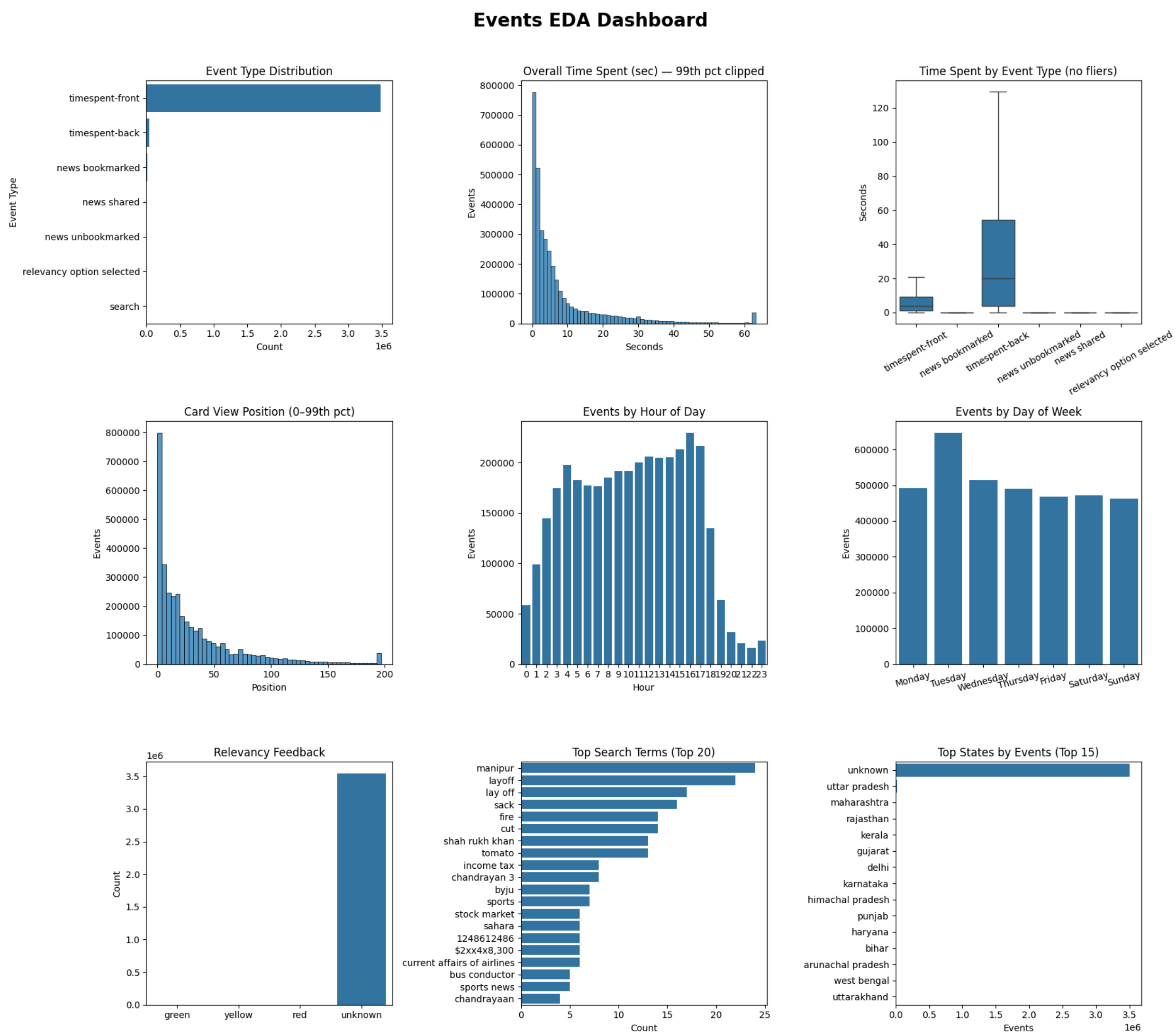


- **Event**

- Event Distribution
  - `timespent-front` dominates (~3.6M events).
  - Other events (`timespent-back`, bookmarks, shares, searches) are negligible in comparison.
- Time Spent
  - Majority of events have <10 sec time spent.
  - `timespent-back` shows the highest median and variability.
  - Other event types contribute little to overall time.
- Card View Position
  - Most engagement happens in the top 20 card positions.
  - Very steep drop-off after position 50; minimal activity beyond 100.
- Temporal Trends
  - Hourly: Peak activity between 9 AM – 9 PM, low activity overnight (12–4 AM).
  - Weekly: Highest on Tuesday, lowest on weekends (Sat & Sun).
- Relevancy Feedback
  - Overwhelmingly labeled as “unknown” (~3.5M).
  - Very few `green`, `yellow`, `red` labels → feedback data is sparse.
- Search Terms
  - Top searches: manipur, layoff, fire, Shah Rukh Khan, income tax, Chandrayaan 3, stock market, sports.

◆ Geographic Distribution

- Large portion marked as “unknown” location (~3.5M).
- Among known: Uttar Pradesh, Maharashtra, Rajasthan, Kerala, Gujarat, Delhi dominate.



- ## Algorithms

Both approaches rely on **embedding-based retrieval** using FAISS. The main difference lies in the **search space** during validation.

Events table is **split chronologically** using `eventTimestamp`:
- **Train Set**: 80% (older events).
- **Validation Set**: 20% (most recent events).

🔹 Approach 1 – Full Content Embeddings

- **Embedding Space**: All content (train + validation).
- **User Representation**:
    - For each user, select **Top-K news interacted with**, based on interaction score.
    - Compute **weighted average embedding** of these Top-K items.
- **Validation Process**:
    - For each user in the validation set, retrieve **nearest neighbours from all available content**.
    - Re-rank candidates using **recency**.

🔹 Approach 2 – Validation-Only Embeddings

- **Embedding Space**: Only validation content.
- **User Representation**: Same as Approach 1 (Top-K weighted average).
- **Validation Process**:
    - Nearest neighbours are retrieved only from the **validation set** (i.e., news available at that time).
    - Re-ranked with **recency**.

3. Algorithm Components

Data Preprocessing

- **Custom CSV Parser** developed to handle **multilingual content** and **emoticons**, ensuring correct tokenization and embedding input.

Interaction Scoring

- Each user's **Top-K news** are chosen based on a **composite score**:
  - **Event Type:** Higher weight for strong signals (e.g., bookmarks > clicks).
  - **Time Spent:** Longer reading time indicates higher interest.
  - **Relevancy Color Feedback:** Explicit signals (green > yellow > red).

Cold Start Strategy

- For new users with no history:
  - Recommend Top-K **popular and recent** news articles.
- Ensures recommendations even for first-time users.

Embedding Generation

- **Multilingual model** `multilingual-e5-base` used to cover diverse language inputs.
- News embedding = **weighted average** of:
  - Title embedding.
  - Content embedding.
  - Metadata embedding.

Vector Indexing

- Embeddings are **L2-normalized**.
  Stored in **FAISS index** for efficient nearest neighbour retrieval.
- Supports large-scale retrieval in milliseconds.

Reranking

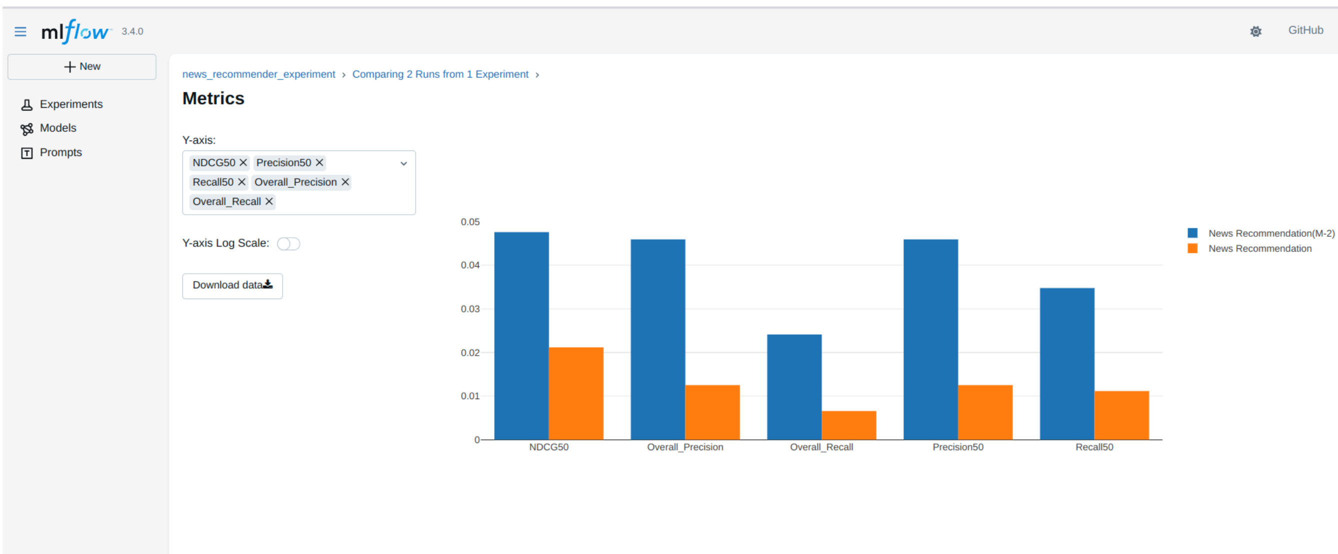
- After FAISS retrieval, candidate news are **re-ranked based on recency**.

Experiment Tracking:

- All runs logged with **MLflow** (embeddings, metrics, parameters).

- # Results

Orange is approach 1 and Blue is approach 2



5. Productionization Plan

User-Specific Workflow

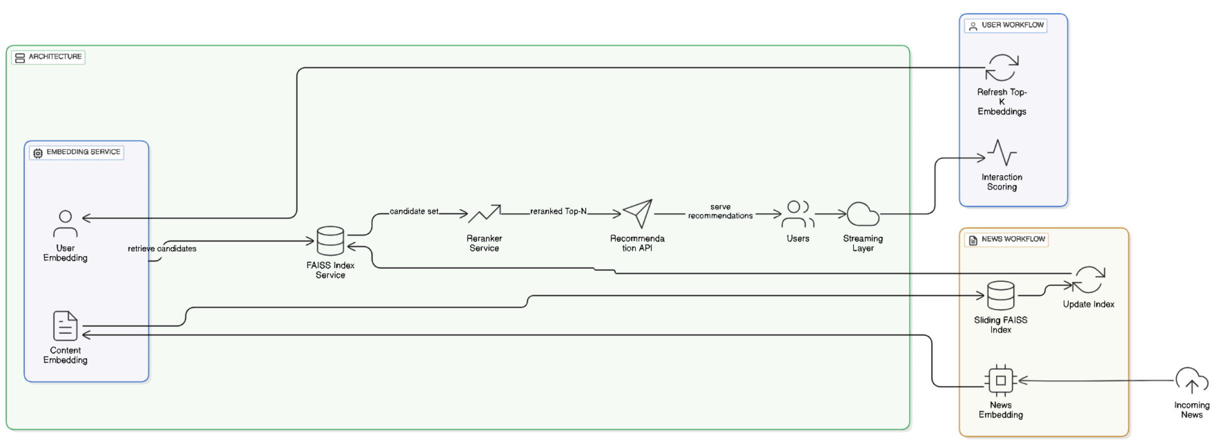
- Generate **interaction score** in real time whenever a user engages with news.
- Refresh **Top-K embeddings per user** every hour (or custom interval).
- Update **recommendations every 12 hours** with latest content.

News-Specific Workflow

- Generate **embeddings for incoming news in real time**.
- Maintain a **sliding FAISS index** of last 1 month's news.
- Update index hourly (or custom interval) to ensure freshness.

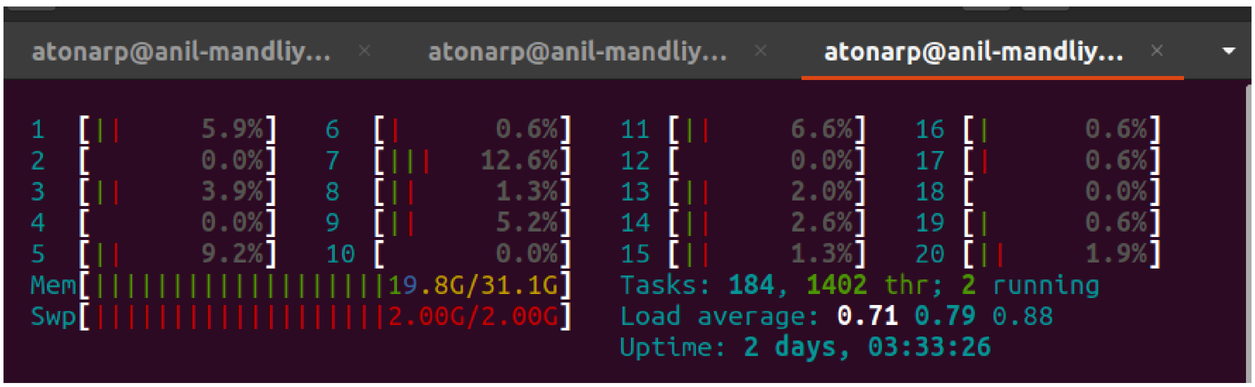
Architecture at Scale

1. **Streaming Layer:** Collect user interactions
2. **Embedding Service:** Generate/update embeddings (user + content).
3. **FAISS Index Service:** Maintain and query nearest neighbour index.
4. **Reranker Service:** Re-rank retrieved candidates with recency + popularity.
5. **Recommendation API:** Serve Top-N news to users.



- **Ending notes:**

- I have considered python to be equivalent to SQL queries and it's not like I am not confident with SQL. its just I didn't spend much time to think on that
- Code requires atleast 20 GB so running on atleast 32 GB RAM laptop is preferable
- Generating embedding is the most resource and time consuming task so atleast 12 core is preferable to optimize runtime



- I have thought of many future directions to optimize the solution but I don' t have much time so leaving it as it is
- ChatGPT proved to be a very helpful tool in completing this assignment. I have used ChatGPT for following things:
  - Understanding domain
  - Generating boilerplate code