YouTube Recommendation System — Technical Architecture Diagram

This document contains a clean, professional architecture diagram and supporting notes for a backend-only YouTube-style recommendation system (data collection → modeling → evaluation → monitoring). Use this as a blueprint for implementation or as a slide/portfolio artifact.

---

1. High-level Components

- **Data Sources**: YouTube Data API, Public Interaction Datasets (e.g., MovieLens-style simulators), Optional: Third-party metadata providers.
- **Data Ingestion**: API fetchers, rate-limit handlers, incremental collectors, scheduler (cron / Airflow).
- **Storage**:
  - Raw storage: CSV / Cloud Storage (GCS / S3) for raw dumps.
  - Processed storage: PostgreSQL / SQLite for metadata; Parquet files for analytics.
- **Processing & Feature Engineering**:
  - Batch ETL jobs (Pandas / PySpark) to clean, normalize, and extract features.
  - Text pipelines: tokenization, stopword removal, TF-IDF, and dense embeddings (USE / BERT).
  - Feature store (simple): parquet or SQL tables for reuse.
- **Modeling**:
  - Content-based models: TF-IDF + Cosine similarity; Embedding + FAISS for ANN.
  - Collaborative models: Matrix Factorization (SVD/NMF), LightFM, or implicit feedback MF.
  - Hybrid layer: weighted ensemble of content and collaborative scores.
- **Evaluation**:
  - Offline metrics: Precision@K, Recall@K, MAP, NDCG.
  - A/B framework for offline experiments (split by users or time windows).
- **Explainability & Analysis**:
  - SHAP/LIME for model explainability on top recommendations.
  - Performance dashboards (Plotly / Matplotlib) for model metrics and dataset stats.
- **Monitoring & Ops**:
  - Data drift detection (feature distributions over time).
  - Model retraining cadence & pipeline orchestration (Airflow / Prefect).
  - Logging & alerts (email / Slack) for pipeline failures.

---

2. Suggested Tech Stack

- Python (Pandas, NumPy)
- NLP: NLTK / SpaCy, TensorFlow Hub (USE) or Hugging Face Transformers
- Modeling: Scikit-learn, Surprise, LightFM, Faiss

- Storage: PostgreSQL (metadata), Parquet files (features), optional S3/GCS
- Orchestration: Apache Airflow or Prefect (cron for simple schedules)
- Experiment tracking: MLflow or simple CSV logs
- Explainability: SHAP
- Visualization: Plotly, Matplotlib (for static), Jupyter Notebooks for results

---

3. ASCII Architecture Diagram

```
 [YouTube Data API]    [User Interaction Datasets]
          |                        |
          v                        v
    [Ingestion Jobs (Python + Scheduler)]
                   |
                   v
        [Raw Storage (CSV / GCS / S3)]
                   |
                   v
       [ETL / Feature Engineering Layer]
            |              |         |
            v              v         v
  [Text Embeddings]  [Numeric Features] [Category Encodings]
            \              |         /
             \             |        /
              v            v       v
          [Feature Store - Parquet / SQL]
                        |
                        v
                 [Modeling Layer]
             /          |           \
   [Content-based] [Collaborative]  [Hybrid Combiner]
       (TF-IDF / USE)   (SVD / LightFM)   (Weighted Score)
                        |
                        v
                 [Offline Evaluation]
                        |
                        v
                  [Reports / Notebooks]
                        |
                        v
           [Monitoring & Retraining Scheduler]
```

4. Implementation Notes & Next Steps

1. **Start small:** collect 10k–20k videos early to iterate quickly.
2. **Prototype content-based model first** (fast to implement and explainable).
3. **Simulate user interactions** if you don't have real watch data — generate realistic session sequences for collaborative filtering experiments.
4. **Use FAISS** when your embedding index grows (50k+ vectors) for fast nearest-neighbor search.
5. **Document every experiment** (hyperparameters, dataset snapshot, evaluation results) to include in your project report.