**Full Review of FAISS-Based Movie Recommendation System**

**Project Overview & Initial Setup**

**Goal:**

* Build a **scalable, efficient movie recommendation system** using:
  + **Collaborative Filtering (FAISS)**
  + **Content-Based Filtering (TF-IDF & FAISS)**
  + **Hybrid Recommendation (Combining Both)**
* Ensure **fast searches**, **memory efficiency**, and **scalability** as the dataset grows.

**Tools & Technologies Used**

|  |  |
| --- | --- |
| **Tool** | **Purpose** |
| **FAISS (Facebook AI Similarity Search)** | Fast Approximate Nearest Neighbors (ANN) for large datasets |
| **Scikit-Learn (TF-IDF Vectorizer)** | Convert movie descriptions into numerical vectors |
| **Pandas & NumPy** | Data manipulation |
| **HDF5 & Pickle** | Model storage and retrieval |
| **AWS S3** | Cloud storage for models and datasets |
| **Streamlit** | Web UI for user interaction |
| **EC2 GPU Instance (g5.2xlarge)** | Training FAISS index efficiently |

**Data Preprocessing & Model Training**

**Key Steps**

|  |  |  |
| --- | --- | --- |
| **Step** | **What We Did** | **Why It Matters** |
| **Cleaned & Remapped movieId** | Ensured **sequential IDs** | FAISS requires **consistent**indices |
| **Removed Duplicate Movies** | Merged **movies with same title but different genres** | Prevented duplicate recommendations |
| **Stored TF-IDF Matrix in csr\_matrixFormat** | Kept sparse for **efficient storage** | Reduced RAM usage |
| **Trained FAISS Index (IndexIVFPQ)** | Used **PQ compression & clusters** | Enabled **fast, scalable search** |
| **Saved Model Files Efficiently** | Stored in **Pickle, FAISS, and HDF5 formats** | Made loading **faster** |

**Optimizing FAISS Training & Memory Usage**

**Key Challenges & Solutions**

|  |  |  |
| --- | --- | --- |
| **Issue** | **Solution** | **Impact** |
| **Large FAISS Index (350MB+)** | Used **memory-mapped FAISS (mmap=True)** | ✅ Reduced RAM usage |
| **FAISS Crashes on Training (OOM Errors)** | Trained in **batches & used GPU temp memory** | ✅ Prevented GPU memory overload |
| **CUDA Out of Memory (GPU OOM)** | Set setTempMemory(1GB) & trained in **iterations** | ✅ Made training feasible |
| **Slow FAISS Search on Large Dataset** | Reduced NUM\_CLUSTERS to **1800**, adjusted nprobe | ✅ Faster search |
| **Inefficient Loading of Model Files** | Used **lazy loading & caching (st.cache\_resource)** | ✅ Prevented redundant memory use |

**Efficient Model Loading & Deployment**

**Key Optimizations**

|  |  |
| --- | --- |
| **Optimization** | **Why It Works?** |
| **Memory-mapped FAISS (mmap=True)** | Prevents excessive RAM usage as FAISS database grows |
| **Lazy loading (if \_var is None)** | Ensures models & data are only loaded when needed |
| **Singleton Pattern for Movie Data & TF-IDF Matrix** | Prevents **repeated loading** of large files |
| **TF-IDF Matrix Stored as Sparse (csr\_matrix)** | Saves memory without impacting performance |
| **Used AWS S3 for Model Storage** | Ensures **scalability & remote access** |

**FAISS-Based Recommendation Implementation**

**Core Recommendation Functions**

|  |  |
| --- | --- |
| **Function** | **What It Does** |
| recommend\_content\_based(title, num\_recommendations=10, min\_similarity=0.4) | Uses FAISS to find **similar movies** based on TF-IDF |
| recommend\_collaborative(user\_id, num\_recommendations=5) | Uses **latent factor models** to suggest movies to a user |
| recommend\_hybrid(title, user\_id=None, content\_weight=0.5, collab\_weight=0.5, num\_recommendations=10) | Combines **content-based & collaborative filtering** for personalized recommendations |

**Optimized FAISS Search Implementation**

query\_vector = movie\_vectors[idx].toarray().reshape(1, -1).astype('float32')

distances, recommended\_indices = faiss\_index.search(query\_vector, num\_recommendations + 1)

✔ **Converts sparse TF-IDF row to dense (.toarray())**  
✔ **Ensures FAISS gets correct vector shape (.reshape(1, -1))**

**Deployment: Running the App on Streamlit**

**Steps Taken**

✔ **Integrated Streamlit for UI**  
✔ **Used st.cache\_resource to optimize model loading**  
✔ **Deployed FAISS & TF-IDF models on AWS EC2**  
✔ **Stored large models on S3, downloading only when needed**

**Streamlit UI Flow**

1️⃣ **User selects a movie** → Calls recommend\_content\_based()  
2️⃣ **User enters a user ID** → Calls recommend\_collaborative()  
3️⃣ **Hybrid model runs if both movie & user ID are provided**

**Final Optimizations & Takeaways**

**Best Practices Implemented**

|  |  |
| --- | --- |
| **Best Practice** | **Why It Works?** |
| **Used Memory-Mapped FAISS** | Reduces RAM usage for large indexes |
| **Stored TF-IDF as Sparse (csr\_matrix)** | Saves memory without impacting performance |
| **Lazy Loaded Models (if \_var is None)** | Prevents reloading & speeds up execution |
| **Cached Models (st.cache\_resource)** | Avoids unnecessary recomputation |
| **Trained FAISS in Batches** | Prevents GPU/CPU overload |
| **Set setTempMemory(1GB) for FAISS** | Prevents **CUDA out-of-memory errors** |

**What This Means for Your System**

✔ **Scalable** → Works with **millions of movies** without excessive RAM usage.  
✔ **Fast** → FAISS enables **real-time search & recommendations**.  
✔ **Optimized** → Uses **batch training, memory mapping, and sparse storage**.  
✔ **Deployed** → Accessible via **Streamlit & AWS**.

**Final Thoughts: What We Achieved**

🔹 **Built a fully functional, scalable movie recommendation system**  
🔹 **Optimized FAISS indexing, training, and searching for large datasets**  
🔹 **Ensured fast recommendations using a hybrid (content + collaborative) approach**  
🔹 **Deployed the app using Streamlit & AWS**