**Phased Action Plan – Movie Recommendation Optimizer**

**Step 1: Data Engineering & Preparation**

* **1a. Data Collection**
  + Fetch metadata from TMDB API: title, genres, cast, crew, overview, ratings, popularity, streaming providers.
  + Pull IMDb ratings and Rotten Tomatoes scores.
  + Supplement with Kaggle/IMDb datasets for collaborative filtering.
* **1b. Data Cleaning**
  + Handle missing values and normalize scores across platforms.
  + Deduplicate movies across TMDB/IMDb/Rotten Tomatoes.
  + Standardize genre taxonomy (e.g., “Sci-Fi” vs. “Science Fiction”).
  + Merge streaming availability with metadata.

For **Step 1b (Cleaning)**, run it in **6 compact phases**:

1. **Schema & Types**  
   Cast columns, parse dates, unify booleans/ints/floats; keep raw vs normalized separation.
2. **ID Resolution & Deduping**  
   Resolve tconst, tmdbId, movieId, rt\_id; drop dupes using stable IDs or (title, year) fallback. Add provenance flags.
3. **Score Normalization**  
   Standardize IMDb/MovieLens/RT/TMDB scores onto consistent scales and validate ranges.
4. **Genres & Taxonomy**  
   Canonicalize genre labels (e.g., “Sci‑Fi” → “Science Fiction”); split to arrays and prep multi‑hot encoding.
5. **Streaming Providers (US default)**  
   Normalize TMDB watch‑provider IDs to friendly names; store providers\_flatrate/rent/buy/ads/free with region + timestamp.
6. **QA & Report**  
   Run non‑null key checks, join coverage, score min/max, sample spot‑checks; update docs/step1b\_report.md.

**Step 2: Feature Engineering**

**Step 2a: Text Features — Sub-Phases**

* **2a.0 Input Audit & Setup**  
  Verify text fields available (overview, synopsis, consensus, reviewText, labeled review corpus). Snapshot schemas.
* **2a.1 Text Cleaning & Normalization**
  + Lowercasing, unicode normalization (NFKC).
  + Strip punctuation, HTML tags, duplicates.
  + Handle missing values (mark as "unknown\_text").
* **2a.2 Feature Families (This was divided into two parts, reflected in progress document)** 
  + **TF-IDF Vectors**: For baseline similarity (overview/synopsis).
  + **Embeddings (BERT / Sentence-Transformers)**: Dense semantic vectors for richer features.
  + **Sentiment / Topic Signals** (optional, from labeled review corpus).
* **2a.3 Index & Storage**
  + Store vectors in data/features/text/.
  + Split into: movies\_text\_tfidf.parquet, movies\_text\_bert.parquet.
  + Enforce canonical\_id as index.
* **2a.4 QA & Report**
  + Validate 100% coverage (no missing embeddings).
  + Sanity checks: cosine similarities behave as expected.
  + Update docs/step2a\_report.md with coverage, sample nearest neighbors.
* **2b. Genre & Crew Features**:

**2b.1 – Genres**

* Extend multi-hot encoding beyond top 20 → cover all 29 canonical genres.
* Output: movies\_genres\_multihot\_full.parquet (87,601 × 29).

**2b.2 – Crew Extraction & Encoding**

* Pull crew (actors, directors) from IMDb.
* Identify top 50 actors + top 50 directors (by frequency).
* Multi-hot encode into binary features.
* Outputs:
  + movies\_actors\_top50.parquet
  + movies\_directors\_top50.parquet

**2b.3 – Index & Storage**

* Consolidate all categorical features (genres + crew) into a single aligned index with canonical\_id.
* Place in data/features/categorical/.
* Create manifest JSON documenting schema.

**2b.4 – QA & Report**

* Coverage checks (what % of movies have top actors/directors, distribution of genres).
* Validation of row alignment (87,601 movies).
* Deliver docs/step2b\_report.md + logs.
* **2c. Numeric Features**:

**2c.1 – Numeric Standardization**

* Standardize and normalize numeric variables:  
  • IMDb score  
  • Rotten Tomatoes critic & audience scores  
  • Popularity  
  • Release year
* Apply scaling (e.g., MinMax or z-score) for comparability across features.
* Handle outliers and missing values appropriately.

**2c.2 – Index & QA**

* Align all numeric features to the canonical\_id index (87,601 movies).
* Generate descriptive statistics (min, max, mean, std) per feature.
* Validate ranges (e.g., IMDb 0–10, RT 0–100, year within valid bounds).
* Confirm no NaN/Inf values.
* Deliverables:  
  • data/features/numeric/movies\_numeric\_features.parquet  
  • QA report (docs/step2c\_report.md) with coverage + stats  
  • Execution log (logs/step2c\_phase2.log)

**2d. Platform Features**

**2d.1 – Provider Encoding**

* Extract streaming availability from TMDB (JustWatch integration).
* Encode availability for each provider (Netflix, HBO, Paramount, Hulu, Prime, Disney+, etc.).
* Use binary/multi-hot encoding across categories: flatrate, rent, buy, ads, free.
* Save as data/features/platform/movies\_platform\_features.parquet.

**2d.2 – Index & QA**

* Validate row alignment (87,601 movies, canonical\_id index).
* Verify all expected providers included and feature dtypes = int8 binary.
* Produce coverage stats (e.g., % of movies available on each platform).
* Create descriptive plots (bar charts of provider distribution).
* Deliverables:  
  • docs/step2d\_report.md  
  • logs/step2d\_phase2.log  
  • Visuals in docs/img/
* **Output**: Movie feature vectors that represent each film across text, categorical, numeric, and platform dimensions.

**Step 3: Recommendation Models**

* **3a. Content-Based Filtering**
* **3a.1 – Feature Matrix Assembly**

**Goal:** Build a single, aligned feature space per movie with sensible weights.

* **3a.2 – Similarity Computation (Cosine + kNN)**

**Goal:** Precompute nearest neighbors for each movie.

* **3a.3 – QA & Spot Checks**

**Goal:** Prove it works and is sane.

* **3a.4 – Documentation & Hand-off**

**Goal:** Package results for downstream use (3b/3c + UI).

* **3b. Collaborative Filtering**

**3b.1 – Ratings Matrix Assembly**

* Build user–movie ratings matrix from IMDb/Kaggle datasets.
* Align on canonical\_id.
* Handle sparsity (drop ultra-rare users/movies if needed).
* Save in both sparse (CSR) and parquet formats.

**3b.2 – Matrix Factorization (SVD/ALS)**

* Train collaborative filtering model (e.g., SVD, ALS).
* Produce latent factors for users and movies.
* Store embeddings and reconstruction metadata.

**3b.3 – Evaluation & Sanity Checks**

* Compute RMSE / Recall@K on held-out ratings.
* Check factor coverage (no NaN, norms within bounds).
* Spot-check a few movies: nearest neighbors by latent space.

**3b.4 – Documentation & Hand-off**

* Final report (docs/step3b\_report.md) with model settings, eval metrics, QA results.
* Update README in data/collaborative/.
* Confirm artifacts are ready for 3c (Hybrid).
* **3c. Hybrid System**

3c.1 – Hybrid Assembly & Alignment  
3c.2 – Candidate Generation & Re-ranking  
3c.3 – Tuning & Offline Evaluation  
3c.4 – Documentation & Hand-off

* **3d. Filtering Layer**
* 3d.0 – Readiness Gate (Preflight)
* 3d.1 – Scoring Service (Stateless “recommend()”)
* 3d.2 – Candidate Fetcher & Cache
* 3d.3 – Shadow Replay (Correctness & Latency)
* 3d.4 – Experiment Design (A/B Config & Assignment)
* 3d.5 – Telemetry & Schemas
* 3d.6 – Monitoring & Dashboards
* 3d.7 – Safety, Rollback & Kill Switch
* 3d.8 – Staging Dry-Run & Launch Checklist

**Step 4: Evaluation & Validation**

 **4.1 – Offline Evaluation (Quantitative Metrics)**

**4.1.1 – Metric Framework Setup**

* Define which metrics to compute: Recall@K, Precision@K, MAP@K, NDCG, RMSE (for CF).
* Set K values (5, 10, 20, 50).
* Confirm ground-truth dataset split strategy (holdout, sampled users).

**4.1.2 – Content-Based Evaluation**

* Evaluate the Step 3a embeddings + kNN outputs.
* Compute Recall@K and MAP@K using MovieLens/IMDb ratings.
* Document coverage and bias (sparse vs dense movies).

**4.1.3 – Collaborative Filtering Evaluation**

* Evaluate Step 3b matrix factorization results.
* Compute RMSE, Recall@K, MAP@K.
* Check factor quality vs user activity levels.

**4.1.4 – Hybrid Model Evaluation**

* Evaluate hybrid scoring (Step 3c, incl. α grid and bucket-gate logic).
* Compare against baselines (content-only, CF-only).
* Compute lift vs baselines.

**4.1.5 – Stratified Analysis**

* Break down metrics by user cohorts: cold, light, medium, heavy (even if cold data is missing, note limitation).
* Break down by movie type (popular vs long-tail).

**4.1.6 – Reporting & Visuals**

* Aggregate results into tables/plots (bar charts for Recall@K, line plots for α sensitivity).
* Save report in docs/step4\_eval\_metrics.md.
* Include acceptance gate checks (e.g., Recall@10 > baseline, coverage ≥ 60%).

 **4.2 – Case Studies (Qualitative Examples)**

**4.2.1 — Case Slate & Sampling Plan**

**Goal:** Define who and what we will inspect, and why.  
**Inputs:** policy\_step4.json, 4.1 results, user activity stats, popularity buckets.  
**Deliverables:**

* data/cases/users\_case\_slate.csv (min 12 users: 4 cold\_synth, 4 medium, 4 heavy; note missing light)
* data/cases/anchors\_case\_slate.csv (≥10 anchor movies covering head/mid/long-tail)
* docs/cases/case\_hypotheses.md (what each case should demonstrate)  
  **Acceptance gates:**
* Users and anchors meet counts and coverage (cohorts × popularity)
* Two “surfaces” defined (e.g., provider/year filter + baseline)

**4.2.2 — Snapshot Generation (Side-by-Side Lists)**

**Goal:** Capture top-K for each system to compare qualitatively.  
**Inputs:** scorer + candidates entrypoints, bucket-gate policy, K=10.  
**Deliverables:**

* data/cases/snapshots/{case\_id}\_{system}.json for systems: content, cf, hybrid\_bg
* docs/img/cases/{case\_id}\_triptych.png (side-by-side top-10 per system)
* logs/step4\_cases\_snapshots.log  
  **Acceptance gates:**
* 100% of cases have valid top-10 snapshots for all systems
* Provenance recorded: alpha used, overrides triggered, filters applied

**4.2.3 — Rationale Attribution & Evidence**

**Goal:** Explain “why recommended” with concrete signals.  
**Inputs:** embeddings/similarity (3a), CF neighbors/factors (3b), rerank/provenance (3c), policy\_step4.json.  
**Deliverables:**

* data/cases/attributions/{case\_id}.json with fields per item:
  + content\_signals (genre overlap, cosine, keywords),
  + cf\_signals (neighbor items/users),
  + policy\_path (alpha chosen, overrides),
  + mmr/diversity notes, provider/year matches
* docs/cases/why\_templates.md + docs/cases/{case\_id}\_why.md (human-readable rationales)  
  **Acceptance gates:**
* Rationale coverage for **100% of hybrid items** in each case
* At least one concrete evidence field per rationale (e.g., “cosine=0.87, shares 3 genres”)

**4.2.4 — Red-Team & Error Taxonomy**

**Goal:** Identify common failure modes with reproducible steps and fixes.  
**Focus modes:** popularity bias, redundancy, stale/irrelevant, provider mismatch, over-sequels, niche misfires, long-tail starvation, cold-start miss.  
**Deliverables:**

* docs/step4\_error\_taxonomy.md (definitions, symptoms, examples, mitigations)
* data/cases/error\_backlog.json (case\_id, trigger, repro recipe, severity, proposed fix, owner)  
  **Acceptance gates:**
* ≥5 distinct failure modes documented with **repro recipes**
* Each failure has a concrete mitigation proposal (e.g., λ\_div tweak, recency boost, stricter provider filter, alpha cap)

**4.2.5 — Policy Validation & Override Tuning**

**Goal:** Verify bucket-gate + overrides behave as intended; propose precise adjustments if needed.  
**Inputs:** policy\_step4.json, case evidence.  
**Deliverables:**

* docs/policy\_step4\_case\_findings.md (where overrides triggered, correctness, edge cases)
* data/hybrid/policy\_step4\_proposals.json (optional tweaks: α\_map, long-tail thresholds, min-history guardrail value)
* docs/policy\_step4\_proposals\_diff.md (diff vs current policy)  
  **Acceptance gates:**
* Pass/fail call for each override encountered (correct/incorrect)
* If proposing changes, include measurable trigger thresholds and rationale

**4.2.6 — Stakeholder Pack & Sign-Off**

**Goal:** Package the qualitative evidence into a decision-ready bundle.  
**Deliverables:**

* docs/step4\_case\_studies.md (executive bullets, case cards, triptychs, rationales, error taxonomy summary, policy check outcomes)
* docs/img/cases/ all visuals referenced in the doc
* docs/step4\_case\_checklist.md (QA checklist + sign-off block)  
  **Acceptance gates:**
* All referenced images/files exist and render
* Clear recommendation statement and sign-off section completed

 **4.3 – Robustness Checks (Edge Cases)**

Focus: stress-test the system in unusual or adversarial conditions.  
**Sub-steps**:

* **4.3.1 – Edge Case Definition & Setup**  
  List edge scenarios (e.g., no providers available, all ratings identical, extreme cold-start, only long-tail anchors, sequel-heavy catalogs).
* **4.3.2 – Execution of Edge Cases**  
  Run snapshots/policy against those edge cases, collect outputs.
* **4.3.3 – Analysis & Findings**  
  Document which failure types appear, whether policy/overrides hold up, and summarize results.

 **4.4 – Documentation & Hand-off**

Focus: package everything for downstream users (teammates, stakeholders, or deployment).  
**Sub-steps**:

* **4.4.1 – Consolidated Docs Assembly**  
  Merge 4.1 + 4.2 + 4.3 reports into a single docs/step4\_final\_report.md.
* **4.4.2 – Artifact Inventory & Validation**  
  Produce an artifact manifest (JSON/MD) listing all files, their sizes, paths, and purposes.
* **4.4.3 – Handoff Package Creation**  
  QA checklist, README snippets, sign-off block, reproducibility metadata (seed=42, commit hash, policy version).

**Step 5: Visualization & Reporting**

* **Visuals**
  + t-SNE/PCA clusters of movies.
  + Word clouds of top keywords.
  + Recommendation flowchart: user input → filtering → ranking → output.
* **Report**
  + Document methodology, models, evaluation, visuals, and UI screenshots.
  + Prepare LinkedIn-ready PDF with engaging storytelling.
  + Push all relevant files (codng, report etc) to github repo.

**Step 6: Deployment & Showcase**

* **6a. UI Development**
  + Build interactive app with Streamlit/Gradio.
  + Inputs: genre multi-select, platform dropdown, sorting option.
  + Outputs: movie list with poster, title, AI summary, IMDb & Rotten Tomatoes scores, release year.
* **6b. Deployment**
  + Deploy app on Streamlit Cloud, Hugging Face Spaces, or Heroku.
  + Provide link in GitHub repo and LinkedIn post.
* **6c. Showcase**
  + GitHub repo: Movie\_Recommendation\_Optimizer.
  + LinkedIn post with screenshots of recommendations and a hook like *“Built a movie recommender that beats Netflix browsing.”*