QUAAACK – Q: Question & Quality Framing

Scalable Movie Recommendation System for Large Datasets

> This notebook documents the **Q** stage of QUAAACK: precise problem framing and quality targets before building. Everything here is movie-specific.

Quick Checklist (Q stage)

- Pin down the core movie-rec question and success metrics.
- Declare assumptions about scale, sparsity, latency, and privacy.
- Align on explainability: every rec should show a clear "because you liked …" anchor.
- Capture stakeholders and risks early.

Q1. Problem / Question Statement

- **User need**: help viewers discover movies they will enjoy faster than scrolling through catalogs.
- **Task type**: personalized top-N ranking + similar-movie lookup; cold-start support via content features.
- **Scope**: English-language movies; demo in Streamlit (web + mobile browser); supports signed-in users only.
- **Success question**: Does the recommender beat a popularity baseline on engagement (e.g., CTR or saves) and offline NDCG@10?
- **Hypotheses**:
- H1: Implicit-feedback collaborative filtering improves NDCG@10 over popularity by __% on MovieLens 25M.
- H2: Adding content embeddings (plot/genre/crew) cuts cold-start error and lifts Precision@10 by __%.
- **Constraints**: p95 latency < 300 ms per request; batch retrain daily; comply with dataset licenses and avoid PII.

Q2. Quality & Evaluation Plan

- **Offline metrics**: NDCG@k, HR@k, MAP, coverage, novelty/diversity; cold-start slice metrics.
- **Operational SLOs**: p95 latency <300 ms; index build < X minutes; memory budget ≤ Y GB; freshness ≤ 24h.
- **Ablations**: popularity baseline, CF-only, content-only, hybrid (tunable weights), SVD rank sweeps.
- **Data quality checks**: sparsity levels; user/item frequency cutoffs; chronological splits to avoid leakage.
- **Explainability bar**: every rec surfaces the top contributing liked movie ("because you liked *Inception*").
- **Failure criteria**: block launch if metric deltas vs baseline are below thresholds or SLOs are missed.

Q3. Big Data Considerations (Movies)

- **Datasets**: MovieLens 25M; TMDB metadata for plots/genres/crew; optional IMDb ratings for priors.
- **Scale challenges**: high sparsity; long-tail movies; temporal drift; incremental updates.
- **Efficiency tactics**: implicit feedback matrices; truncated SVD; approximate nearest neighbors (Faiss/Annoy) for similarity; batch scoring for home pages.
- **Storage/layout**: interactions in Parquet; precomputed movie embeddings; shard ANN index by decade/genre if needed.

Q4. Stakeholders, Risks, Ethics

- **Stakeholders**: end viewers; content owners/licensors; product & data teams; infra/ops; compliance/legal.
- **Risks**: popularity bias; cold-start blind spots; filter bubbles; data drift; license misuse.
- **Mitigations**: diversity re-ranking; content backfill for new movies; periodic retrains; license checks; anonymized logs.

Q5. Streamlit UI Hooks (for later phases)

- Movie selector or search box; "similar to this" mode.
- Recommended movies list with scores and short reasons (liked-movie anchors).
- Filters: genre, release year, language, diversity toggle.
- Logging: capture clicks/saves to feed implicit feedback.

Experiment Log Template

Date	Variant	Params	Metric@k	Latency	Outcome

In [1]:
```
# Minimal popularity baseline for sanity checks (replace with MovieLens 25M
slices)
import pandas as pd

movies = pd.DataFrame(
    {
        "item_id": [101, 102, 103, 104, 105, 106],
        "title": [
            "Inception",
            "The Dark Knight",
            "Interstellar",
            "The Matrix",
            "Fight Club",
            "Spirited Away",
        ],
    }
)

# Toy interactions; swap with real data
interactions = pd.DataFrame(
    {
        "user_id": [1,1,2,2,2,3,3,4,4,4],
        "item_id": [101,102,101,103,104,104,105,101,104,106],
        "rating": [1]*10,  # implicit feedback
    }
)

popular = (
    interactions.groupby("item_id").size().sort_values(ascending=False)
    .rename("popularity")
    .reset_index()
    .merge(movies, on="item_id", how="left")
)

def recommend_popular(top_n=5):
    return popular.head(top_n)[["title", "popularity"]]

print(recommend_popular())
```

```
                title  popularity
0       Inception           3
1      The Matrix           3
2  The Dark Knight           1
3     Interstellar           1
4      Fight Club           1
```

In [2]:

```
# Simple "because you liked X" reason generator (movie titles)
import random

user_likes = interactions.groupby("user_id")["item_id"].apply(list)
movie_lookup = dict(zip(movies.item_id, movies.title))

def explain_recommendation(user_id, recommended_item):
    liked_items = user_likes.get(user_id, [])
    anchor_id = random.choice(liked_items) if liked_items else None
    if anchor_id:
        anchor_title = movie_lookup.get(anchor_id, f"movie {anchor_id}")
        return f"Recommended because you liked '{anchor_title}'."
    return "Recommended based on similar viewers and content."

print(explain_recommendation(2, 104))
```

```
Recommended because you liked 'Inception'.
```

In [ ]: