International Journal of Interpreting Enigma Engineers (IJIEE)
https://ejournal.svgacademy.org/index.php/ijiee/
https://doi.org/10.62674/ijiee.2026.v3i02.003
OPEN ACCESS

FILMIQ STUDIO PRO: AI-POWERED MOVIE RECOMMENDATION SYSTEM

Purvi Lakhotia[1], Afra Anees[2], Nandini[3]

Original Article

[123]Undergraduate Student, Department of CSE (AIML), Stanley College of Engineering and Technology for Women, Abids, Hyderabad, India.

*Corresponding Author's Email: lakhotiapurvi@gmail.com[1], afraanees92@gmail.com[2], nandinikarre2529@gmail.com[3]

Abstract

Streaming platforms keep adding more titles every month, and at some point that stops being a good thing. Scroll through any app for ten minutes and you'll see what we mean — rows and rows of posters, half of which mean nothing to you, and no real sense of why any particular movie showed up there. Most search bars are still just keyword matchers underneath, so they're fine if you already know the exact title you want, but pretty useless if what you actually want is "something like the last movie I watched." That's the gap we tried to close with FilmIQ Studio Pro. Instead of matching words, we use Sentence-BERT to turn each movie's overview, genres, cast, director and keywords into one dense vector that captures what the movie is actually about, and then a K-Nearest Neighbors model finds the closest movies to whatever you pick, using cosine similarity. There's no need for ratings history or past clicks — the recommendation is built purely from the movie's own content. Around that core engine we added a few things that made the app genuinely nicer to use day-to-day: mood-based browsing across six moods, a "Surprise Me" button for when you can't decide, a watchlist that persists for the session, and an analytics tab built with Plotly so you can poke around the dataset itself. Everything runs in Python — Streamlit for the UI, Pandas and Scikit-learn underneath, Sentence Transformers for the embeddings, and the TMDB API for poster art. What we ended up with is proof that you don't need a mountain of user data to make recommendations feel personal — sometimes the content itself is enough.

Keywords: Movie Recommendation System, Sentence-BERT, K-Nearest Neighbors, Natural Language Processing, Content-Based Filtering, Streamlit, Semantic Similarity, Mood-Based Filtering.

Introduction

Streaming catalogues today run into tens of thousands of titles, and somewhere along the way the problem flipped on its head. It used to be that people couldn't find enough to watch. Now there's too much, and picking something has become its own little chore. There's actually a name for this in industry research — choice overload — and a surprising number of viewing sessions end with nobody watching anything at all, simply because the person gave up scrolling before deciding. That's the exact itch recommendation systems are supposed to scratch.

Most of the search tools built into these platforms are still doing keyword matching under the hood: type in a title, an actor, maybe a genre, and you get back whatever text matches. That works fine if you already know what you're looking for. It falls apart the moment you don't — because two movies can be totally unrelated on paper (no shared actors, no overlapping words in the description) and still feel like the same kind of movie to a viewer. Keyword search just isn't built to notice that.

That's the problem we set out to solve with FilmIQ Studio Pro, using NLP and a bit of machine learning instead of plain text search. We lean on Sentence-BERT (SBERT) — a transformer model built for sentence-level embeddings — to take each movie's overview, genres, cast, director and keywords and turn all of that into a single dense vector that represents what the movie is actually about, not just the words used to describe it. From there, a K-Nearest Neighbors

model trained on these vectors with cosine similarity lets us pull out movies that are conceptually close to whatever the user picked, even when the two films barely share any vocabulary.

On top of that core engine we layered a handful of features that, honestly, came more from "this would be nice to have" than from any formal requirement: mood-based browsing for when someone wants a vibe rather than a title, a Surprise Me option for the indecisive, a watchlist for the current session, and an analytics tab that shows what the dataset itself looks like. The whole thing runs as a Streamlit app with a dark, cinema-style interface, and posters are pulled live from the TMDB API.

Problem Statement

Streaming platforms keep growing, and so do their catalogues — at this point most of them hold far more movies and shows than anyone could browse through manually. A lot of that browsing time isn't spent watching anything; it's spent just trying to decide what to watch. Keyword-based search doesn't really help here because it has no way of understanding what a movie is actually about beneath the surface-level text. That leaves a real gap for a system that can look at a movie's content and figure out what else is similar to it, rather than just matching words a user happens to type in.

Problem Specification

Put simply: the system needs to recommend movies based on what's in them, not on what other users rated highly. That means working with five kinds of metadata — genres, keywords, cast, directors, and overviews — and using ML/NLP to figure out how movies relate to each other semantically. We set four goals for ourselves going in: the recommendations had to be accurate, fast enough to feel instant, scalable across thousands of titles, and wrapped in an interface that didn't feel like a research demo. Figure 1 lays this out, along with the actual technique used under the hood — SBERT embeddings narrowed down with KNN and ranked by cosine distance.

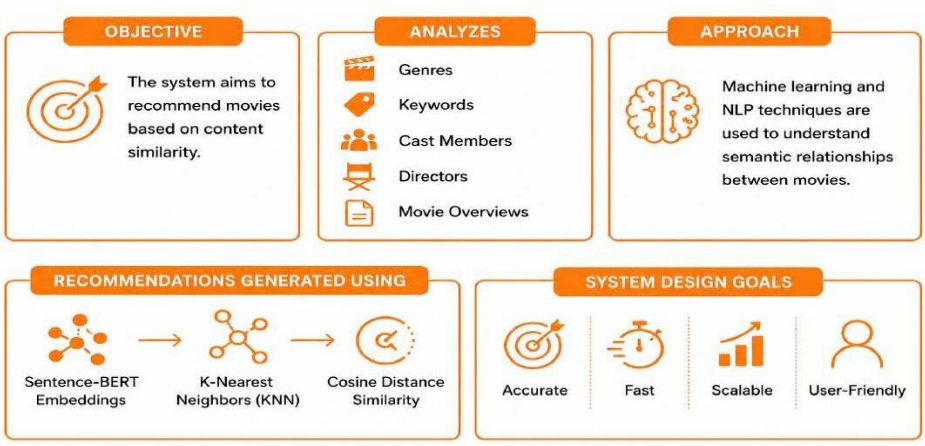

Figure 1: Problem Specification — Objective, Analysis Scope, Approach, and Design Goals

Objectives of the Project

1. Build a content-based movie recommendation system that doesn't rely on user rating history.
2. Generate recommendations based on semantic similarity between movie descriptions rather than overlapping keywords.
3. Add mood-based filtering so people can browse by feeling (Feel Good, Dark, Intense, and so on) instead of just by title.
4. Give users a way to save and manage a watchlist during their session.

Scope of the Project

- Recommends movies by applying NLP and ML techniques to mostly unstructured text fields.
- Uses SBERT embeddings to capture what a movie's overview, genres, cast, director, and keywords actually mean.
- Works off the TMDB 5000 Movies and Credits datasets for all metadata.

- Helps people discover movies even when they don't have a specific title in mind.
- Built for entertainment use cases, with room to plug into OTT or streaming platforms later on.

Applications of the Project

Personalized Movie Recommendations — Suggestions come from actual movie similarity, not just whatever's trending.

Easy Movie Discovery — Quickly surfaces movies similar to ones a user already likes.

Mood-Based Recommendations — Lets people browse by mood — Feel Good, Funny, Dark, Intense, Chill, or Adventurous — instead of hunting for a title.

Watchlist Management — Save movies for later, remove them once watched.

Streaming Platform Integration — The recommendation engine itself could be dropped into an OTT platform as a discovery layer.

Enhanced User Experience — Cuts down on the time spent searching and makes discovery feel less like a chore.

Literature Survey

Before settling on our approach, we went through a fair amount of recent work on movie recommendation systems, mostly to get a sense of what's already been tried and where it tends to fall short. Table 1 pulls together ten papers from 2024 through 2026 — everything from BERT-based deep learning and sentiment analysis to reinforcement learning and multi-modal transformer setups that bring in poster images alongside text.

Literature Survey 1	**Shreya Patil, Ramya R. Bijapur, Anupama Bidargaddi, et al. — "Enhancing Movie Recommendation Systems with BERT: A Deep Learning Approach" (INCET Conference, 2024)**
Problem Taken	Improving recommendation accuracy using semantic understanding
Methodology	BERT-based Deep Learning
Dataset	MovieLens Dataset
Performance Measures	Recommendation Accuracy, Precision
Improvements	Better semantic understanding of movie descriptions
Limitations	High computational cost
Literature Survey 2	**Hyun-Chul Lee, Yong-Seong Kim, Seong-Whan Kim — "Real-Time Movie Recommendation: Integrating Persona-Based User Modeling with NMF and Deep Neural Networks" (Applied Sciences, 2024)**
Problem Taken	Real-time personalized movie recommendations
Methodology	NMF + Deep Neural Networks
Dataset	Movie Recommendation Datasets

Performance Measures	Recommendation Quality, User Satisfaction
Improvements	Persona-based personalization
Limitations	Increased model complexity
Literature Survey 3	**Amany M. Sarhan, et al. — "Integrating Machine Learning and Sentiment Analysis in Movie Recommendation Systems" (Journal of Electrical Systems and Information Technology, 2024)**
Problem Taken	Improving recommendation quality using user sentiments
Methodology	Machine Learning + Sentiment Analysis
Dataset	Movie Reviews and Ratings Data
Performance Measures	Accuracy, Precision
Improvements	Captures user opinions effectively
Limitations	Dependent on sentiment quality
Literature Survey 4	**Qi Wang, YinLi You, Si Wang — "Personalized Movie Recommendations Based on Probabilistic Linguistic Sentiment and DEMATEL-TODIM Methods" (Scientific Reports, 2024)**
Problem Taken	Personalized recommendations using review analysis
Methodology	Sentiment Analysis + DEMATEL-TODIM
Dataset	Online Movie Reviews
Performance Measures	Recommendation Relevance
Improvements	Improved personalization
Limitations	Complex implementation
Literature Survey 5	**Hanyue Mao, Yang Fan, Mingwen Tong — "Research on Aspect-Based Sentiment Analysis of Movie Reviews Based on Deep Learning" (Journal of Information Science, 2024)**
Problem Taken	Understanding user opinions from movie reviews
Methodology	BERT, RoBERTa, Deep Learning
Dataset	Movie Review Dataset

Performance Measures	Classification Accuracy
Improvements	Better sentiment extraction
Limitations	Requires large training data
Literature Survey 6	**Sony Peng, Sophort Siet, Dae-Young Kim, et al. — "Integration of Deep Reinforcement Learning with Collaborative Filtering for Movie Recommendation Systems" (Applied Sciences, 2024)**
Problem Taken	Cold-start and sparsity issues
Methodology	Deep Reinforcement Learning + Collaborative Filtering
Dataset	MovieLens Dataset
Performance Measures	Accuracy, Recall
Improvements	Better adaptive recommendations
Limitations	High training cost
Literature Survey 7	**Yang Gao, Hong Zheng, Haonan Cui — "User Preference Modeling for Movie Recommendations Based on Deep Learning" (Scientific Reports, 2025)**
Problem Taken	Capturing complex user preferences
Methodology	Deep Learning, Graph-Based Methods
Dataset	User Behaviour and Movie Content Data
Performance Measures	Personalization Accuracy
Improvements	Better preference prediction
Limitations	Requires large datasets
Literature Survey 8	**Raghavendra C. K., Srikantaiah K. C., Sunil C. K. — "Dual Module Wider and Deeper Stochastic Gradient Descent and Dropout Based Dense Neural Network for Movie Recommendation" (Scientific Reports, 2026)**
Problem Taken	Personalized movie recommendation in streaming platforms
Methodology	Dense Neural Network with SGD and Dropout
Dataset	Movie Rating Dataset

Performance Measures	Recommendation Accuracy
Improvements	Improved prediction performance
Limitations	Computationally intensive
Literature Survey 9	**Various Researchers — "Movie Recommendation Based on Deep Neural Networks" (Procedia Computer Science, 2025)**
Problem Taken	Personalized movie recommendations
Methodology	Neural Collaborative Filtering (NCF)
Dataset	MovieLens Dataset
Performance Measures	Hit Rate, Accuracy
Improvements	Better recommendation robustness
Limitations	Requires extensive training
Literature Survey 10	**Linhan Xia, et al. — "Movie Recommendation with Poster Attention via Multi-modal Transformer Feature Fusion" (2024)**
Problem Taken	Improving recommendation quality using multiple data sources
Methodology	BERT + Vision Transformer + Transformer Fusion
Dataset	MovieLens Dataset and Movie Posters
Performance Measures	Rating Prediction Accuracy
Improvements	Uses text and poster information together
Limitations	Computationally expensive

Table 1: Summary of Literature Survey

Research Gap

Existing Limitations

- Collaborative filtering runs into the classic cold-start issue — it has nothing useful to say about a new user or a newly added movie.
- Matrix factorization needs a decent amount of user interaction history before it produces anything meaningful, which isn't always available.
- Deep learning models can work well, but they're hungry for large datasets and computing power that aren't always on hand for a project like this.

- A lot of existing systems stop at producing a ranked list and don't really do much with richer interaction features like mood-based browsing or watchlists.

Proposed Solution

FilmIQ Studio Pro gets around most of this by pairing a lightweight but semantically rich representation (SBERT embeddings) with a similarity search structure (KNN) that's cheap to run. Because recommendations come purely from movie content rather than user history, the cold-start problem basically doesn't apply here — a brand-new user gets just as good a recommendation as a returning one. On top of that, mood-based filtering and watchlist management give the system more to offer than just a bare list of suggestions.

Proposed Methodology

The recommendation pipeline runs through seven stages — starting with the raw dataset and ending at the screen the user actually interacts with. Figure 2 lays all seven out, and each one is broken down below.

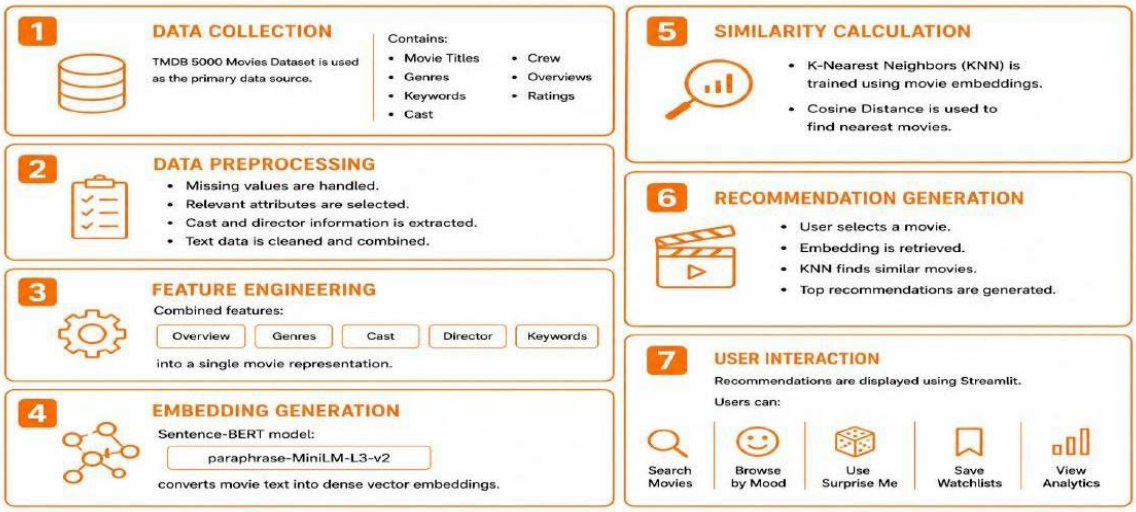

Figure 2: Proposed Methodology — Seven-Stage Recommendation Pipeline

Data Collection

We used the TMDB 5000 Movies dataset along with its companion Credits dataset, which together cover roughly 5,000 films — titles, genres, keywords, cast, crew, overviews, and ratings. The two are merged on a shared movie ID so that cast and crew details line up correctly with each film's descriptive metadata.

Data Preprocessing

A lot of the raw TMDB fields come in as stringified JSON — a list of genre objects stored as plain text, for instance — so each of these columns (genres, keywords, cast, crew) gets parsed with Python's ast.literal_eval and turned into clean lists of names. Anything missing or malformed just falls back to an empty list instead of throwing an error. Director names get pulled out separately from the crew list by looking for whichever entry has "Director" as the job title. Rows that end up with no usable text after all this get dropped, since there's nothing for the embedding model to work with there.

Feature Engineering

Once everything's cleaned up, five fields — overview, genres, the top 5 cast members, director, and the top 8 keywords — get joined into one combined text column. The idea is to give the embedding model the full picture of a movie rather than just its overview, so genre, key cast, directorial style and thematic keywords all factor into the final recommendation.

Embedding Generation

That combined text gets run through a pretrained Sentence-BERT model — paraphrase-MiniLM-L3-v2 — via the Sentence Transformers library. What comes out is a normalized, dense vector per movie that captures meaning rather than exact wording, so movies with a similar feel end up close together in that vector space even if their descriptions don't share much vocabulary. Since generating these embeddings is the slow part, the results (plus the trained KNN

model and a signature for cache invalidation) get saved to disk with joblib so this step only has to run once, not every time the app restarts.

Similarity & Model Development
A K-Nearest Neighbors model from Scikit-learn is trained over the full set of embeddings using cosine distance, keeping up to 50 neighbors per movie. Cosine distance made the most sense here because it cares about the angle between two vectors rather than their length, so it's not thrown off by one overview being longer or shorter than another.

Recommendation Generation
When someone picks a movie, we look up its index in a title-to-index map and pull its embedding, then query the trained KNN model with it. Whatever comes back (minus the movie itself) becomes the recommendation list, each one tagged with a similarity score worked out as (1 − cosine distance) × 100 — basically a percentage "match" — alongside the title's rating, genres, director and overview.

User Interaction
All of this — recommendations, mood-filtered results, analytics — comes together in a Streamlit interface. From there, users can search for a movie and see what's similar, browse by mood, hit Surprise Me for a random pick, manage a watchlist, and look through some basic analytics on the dataset itself.

System Architecture

Figure 3 shows the full system architecture — eight blocks running from the raw TMDB data all the way to what the user sees on screen. Roughly speaking it breaks into a data layer (source, preprocessing, feature engineering), a modelling layer (embeddings plus similarity search, backed by a vector store), an application layer (the five features users actually touch), and finally the presentation layer — the Streamlit interface and whatever it outputs.

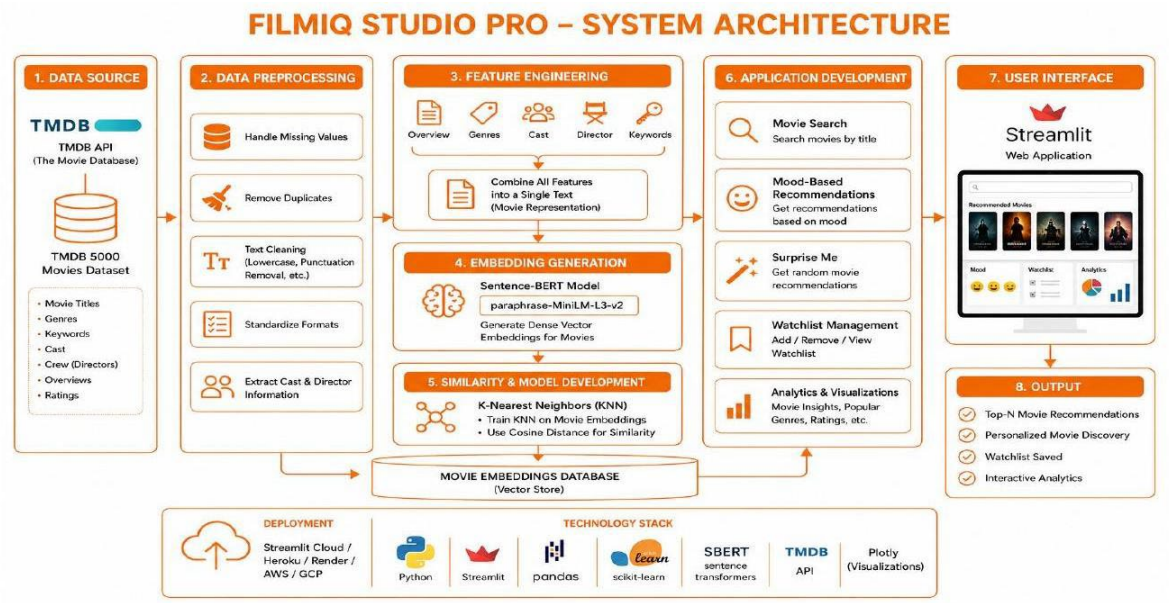

Figure 3: FilmIQ Studio Pro — System Architecture

Workflow
Figure 4 traces the same idea as a single path: the TMDB dataset gets preprocessed and engineered into one text representation per movie, that gets embedded with Sentence-BERT, and a KNN-based similarity search runs on top of those embeddings to power the recommendation engine — which ranks and filters candidates before handing back the final top-N list, alongside whatever watchlist or analytics data the user's working with at the same time.

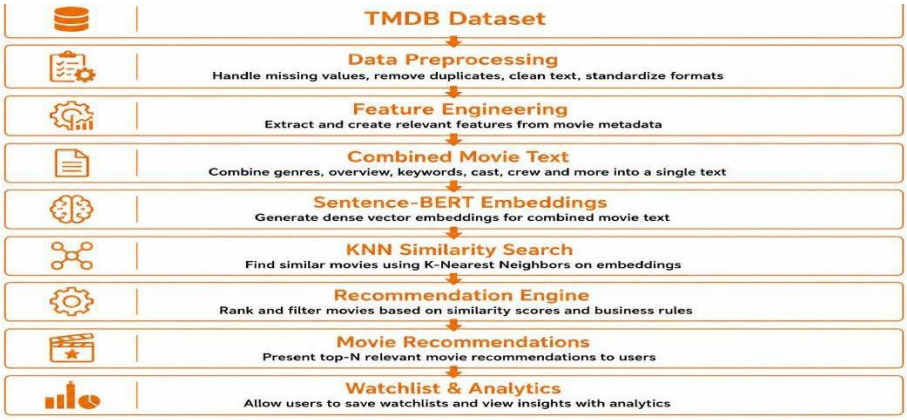

Figure 4: End-to-End Workflow

Code Implementation

Recommendation Model — Feature Combination

Genres, cast, director, keywords and overview all get merged into one text field, combined_text — this is what the embedding model actually sees:

```
df["combined_text"] = (
    df["overview"].fillna("") + " " +
    df["genres_list"].apply(lambda x: " ".join(x)) + " " +
    df["cast_list"].apply(lambda x: " ".join(x)) + " " +
    df["director"].fillna("") + " " +
    df["keywords_list"].apply(lambda x: " ".join(x))
)
```

Embedding Generation

That combined text gets encoded into a dense vector with a pretrained Sentence-BERT model. We normalize the embeddings so cosine similarity ends up being just a dot product, which keeps things fast:

```
model = SentenceTransformer("paraphrase-MiniLM-L3-v2")

embeddings = model.encode(
    df["combined_text"].tolist(),
    convert_to_numpy=True,
    batch_size=64,
    normalize_embeddings=True,
)
```

Recommendation Engine

A K-Nearest Neighbors model gets trained on the embedding matrix with cosine distance, so pulling the nearest movies for any given title is quick:

```
knn = NearestNeighbors(metric="cosine", n_neighbors=50)
knn.fit(embeddings)

distances, indices = knn.kneighbors(
    embeddings[selected_idx].reshape(1, -1), n_neighbors=n+1
)
```

Intelligent Features and User Interaction

Beyond the core engine, there's mood-based filtering (checking movies against favoured/avoided genre lists per mood), a Surprise Me feature that's basically a random sample, and a watchlist that lives in the session state:

```
if st.button("Surprise Me!"):
    pool = df if surprise_mood == "Any mood" else filter_by_mood(df, surprise_mood)
    st.session_state.surprise_pick = pool.sample(1).iloc[0]

st.session_state.watchlist[int(movie_id)] = row_dict
```

Technology Stack

The whole thing is Python end to end. Table 2 lists what we used and what each piece is actually doing in the system.

Technology	Role in the System
Python	Core language for the whole application.
Streamlit	Builds the web interface and handles session state.
Pandas	Loads, merges and manipulates the TMDB movies and credits datasets.
Scikit-learn	Provides the K-Nearest Neighbors model used for similarity search.
Sentence Transformers	Generates the SBERT embeddings (paraphrase-MiniLM-L3-v2) from movie text.
Plotly	Draws the analytics charts — genre counts, rating distribution.
TMDB API	Supplies poster artwork, fetched concurrently and cached to disk.
Joblib	Caches the embeddings, the KNN model, and poster URLs so they don't get rebuilt every run.

Table 2: Technology Stack and Role

Results and Discussion

Application Interface

FilmIQ Studio Pro ships as a five-tab Streamlit app, with each tab handling one piece of the recommendation engine. We went with a dark, cinema-style theme — near-black background, a warm red accent, gold highlights — using the Oswald and Inter fonts, mostly because the default Streamlit look felt a bit too "dashboard-y" for something meant to feel like a movie app.

Search by Movie

In the Search by Movie tab, you pick any title from the dataset and choose how many recommendations you want back, anywhere from 5 to 20. Hit Recommend, and the system pulls the nearest neighbours of that movie's embedding and lays them out as poster cards — each one showing a percentage match score, star rating, and genre tags. There's a watchlist toggle on every card and an expandable overview if you want more detail. You can also export the results straight to CSV from the interface.

Browse by Mood

Browse by Mood lets you pick one of six moods — Feel Good, Intense, Dark, Funny, Adventurous, or Chill — each tied to its own set of favoured and avoided genres plus a minimum rating cutoff. Pick a mood and the dataset gets filtered

down to the top 20 highest-rated matches. If a mood's rules are too strict and turn up nothing for a particular dataset, the filter automatically loosens to just the favoured genres so the page never just sits there empty.

Surprise Me

For anyone who'd rather not pick, Surprise Me grabs a random movie — optionally narrowed to a mood — and shows it with its poster, rating, genres, director and overview. There's a Roll Again button if the first pick doesn't land, and you can add it to the watchlist straight from there.

Watchlist

The Watchlist tab shows whatever the user has saved during the current session, using the same poster-card layout as everywhere else. You can remove single titles or wipe the whole list. Since it's all kept in Streamlit's session state, the watchlist sticks around for as long as the browser tab is open but doesn't carry over between visits — that's something we'd want to fix in a later version.

Analytics Dashboard

The Analytics tab has two Plotly charts built straight from the dataset: a bar chart of the ten most common genres, and a histogram of average ratings across all the movies. Both are styled to match the dark theme, so it doesn't feel like a different app when you switch tabs — it's mostly there to give a sense of what the dataset actually looks like under the hood.

Discussion

Across all five tabs, what stood out to us is that the core idea actually held up — semantic, content-based similarity gives relevant, varied recommendations without needing any rating history at all. Because the engine only looks at movie metadata, it works just as well for obscure titles as it does for the big, well-known ones, and it sidesteps the cold-start problem that trips up collaborative filtering entirely. Caching the embeddings and poster URLs to disk also made a noticeable difference — the app feels snappy on every run after the first.

Planning of the Work

We split the project into six broad stages, starting with requirement analysis and ending with testing and evaluation — Figure 5 lays this out.

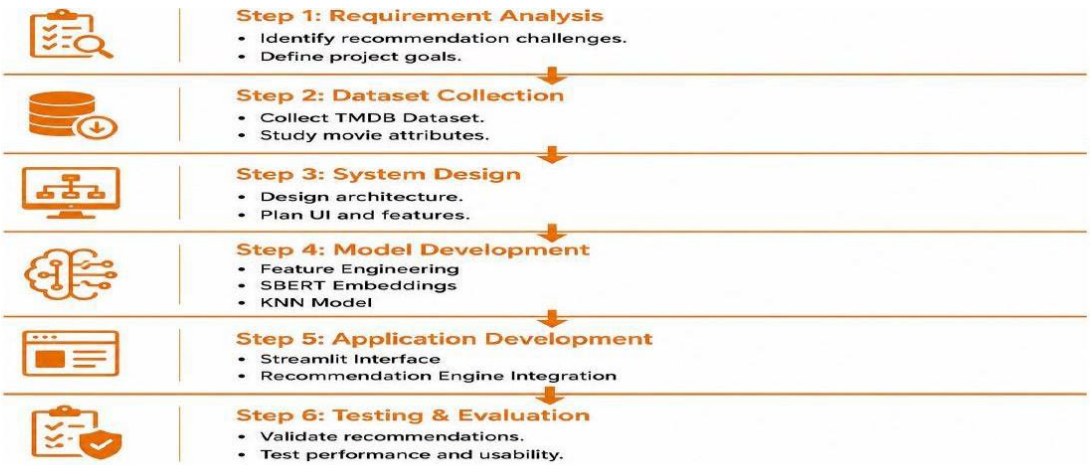

Figure 5: Planning of the Work

Figure 6 zooms into the same plan as a more granular, ten-step build sequence — basically the order we actually worked through while building FilmIQ Studio Pro, from the first requirement discussions through to deployment.

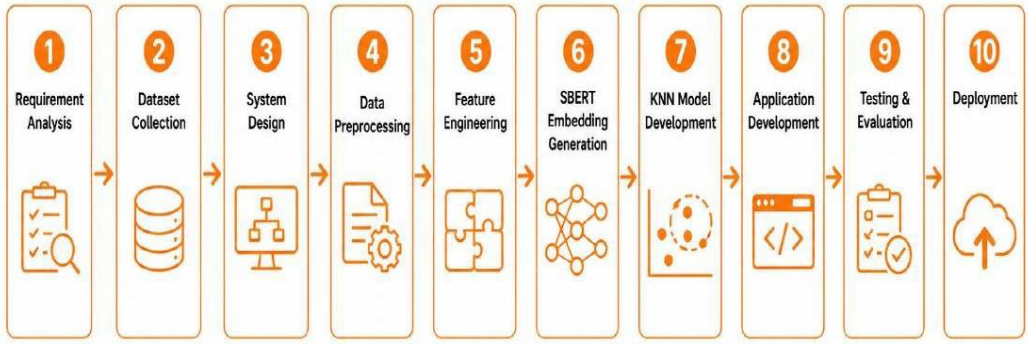

Figure 6: Detailed Development Work Flow

Future Scope and Conclusion

Future Scope

FilmIQ Studio Pro does what it set out to do, but there's plenty of room to take it further. A few things we'd want to explore next:

- Layering in collaborative filtering on top of what already exists, once there's enough real user interaction data to make it worthwhile.
- A hybrid system that blends content-based and collaborative signals instead of relying on just one.
- Persistent profiles, so watchlists and preferences survive across sessions instead of resetting every time the browser closes.
- Ratings that adapt over time based on what someone's actually been watching.
- Keeping the catalogue in sync with TMDB automatically as new movies get added.
- Trying out deeper models — neural collaborative filtering or transformer-based rerankers — to push recommendation quality further at scale.

Conclusion

FilmIQ Studio Pro shows that Sentence-BERT embeddings paired with a KNN similarity search can make for a genuinely useful, content-based movie recommender. By looking at genres, keywords, cast, directors and overviews together rather than separately, the system picks up on relationships between movies that plain keyword matching would completely miss. The mood-based filtering, Surprise Me mode, watchlist, and analytics dashboard round it out into something that feels more like a complete discovery tool than just a recommendation list. And because it doesn't depend on any user history, it stays lightweight enough to run as a single Streamlit app without needing heavy infrastructure behind it.

References

1. TMDB 5000 Movie Dataset. [Online]. Available: https://www.kaggle.com/datasets/tmdb/tmdb-movie-metadata

2. N. Reimers and I. Gurevych, "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks," in Proceedings of EMNLP-IJCNLP, 2019. [Online]. Available: https://arxiv.org/abs/1908.10084

3. F. Pedregosa et al., "Scikit-learn: Machine Learning in Python," Journal of Machine Learning Research, vol. 12, pp. 2825-2830, 2011. [Online]. Available: https://jmlr.org/papers/v12/pedregosa11a.html

4. Streamlit Documentation. [Online]. Available: https://docs.streamlit.io

5. Plotly Documentation. [Online]. Available: https://plotly.com/python/
6. The Movie Database (TMDB) API Documentation. [Online]. Available: https://developer.themoviedb.org/docs
7. Sentence Transformers Documentation. [Online]. Available: https://www.sbert.net
8. Python Software Foundation, Python Documentation. [Online]. Available: https://docs.python.org