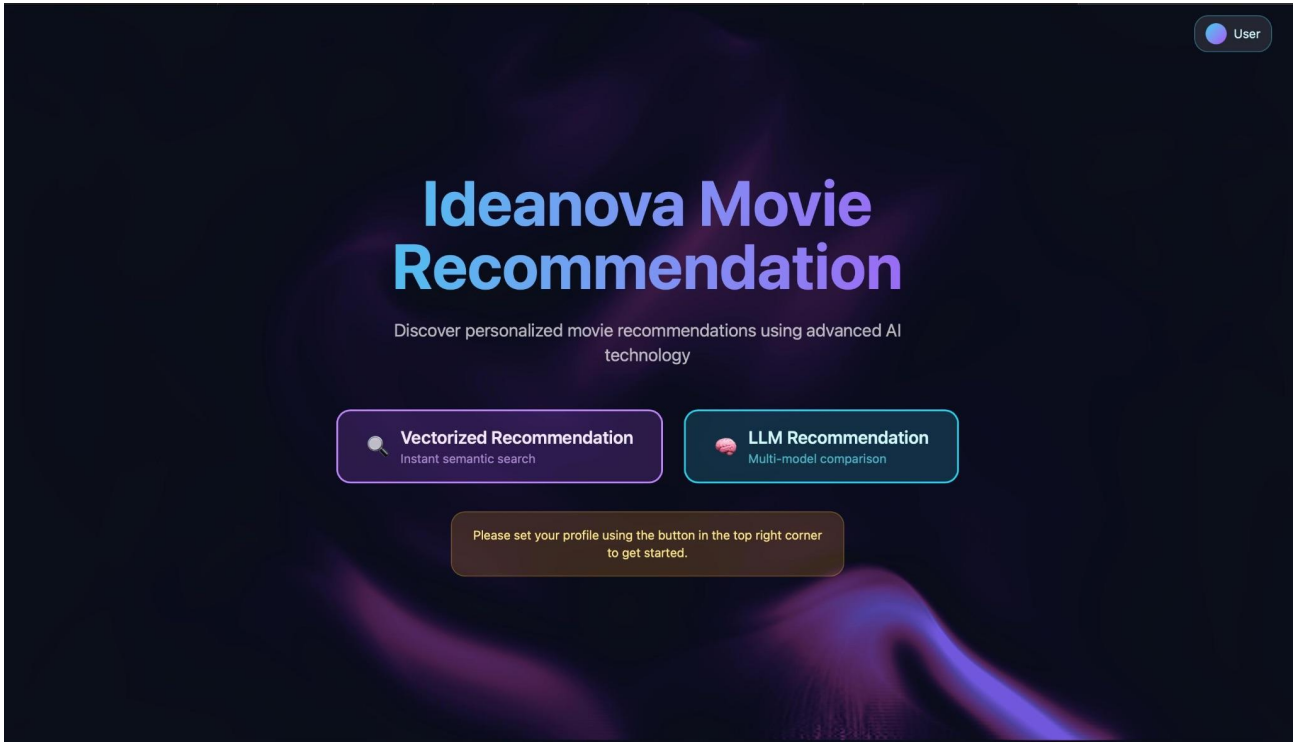
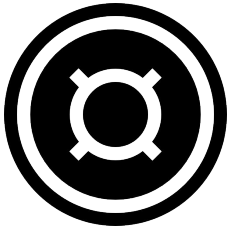
AI Recommendation Engine



–**AI Media Recommendation Engine**, designed to deliver fast, accurate, and explainable media suggestions to users of a streaming platform.

Team 6
Ssai Sanjanna Ganji
Honey Reddy Nagireddy
Siddhartha Gudipudi
Preethi Reddy Nagireddy

WHY - The Problem



Generic recommendations
– All users see similar content



Minimal personalization
– Doesn't consider viewing/shopping history



Poor user experience
– Irrelevant suggestions

HOW - Our Solution

-> We designed a **dual-path recommendation backend** that combines:

- A **semantic vector recommendation engine** for fast, deterministic similarity search
- An **LLM recommendation engine** that enriches the user profile

-> The system generates **high-quality personalized recommendations** without relying on a single technique.

-> Works across **multiple catalogs** (Movies and Shopping) using the same backend architecture.

-> Designed to remain **fully operational even if external AI services are unavailable**.

Tech & Tools Used

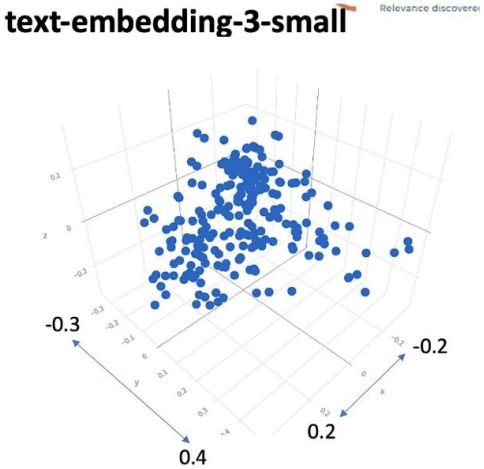








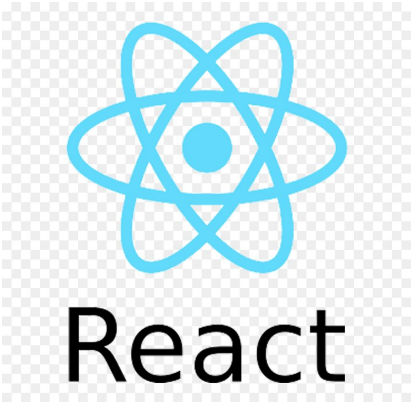
Tech & Tools Used



OpenAI API

Hugging Face

Tech & Tools Used







Vector Recommendation Engine Overview

- Unified vector engine supporting **movies and shopping domains**
- Designed for **real-time, low-latency recommendations**
- Embedding-based similarity search using **cosine similarity**
- Dual embedding support:
  - **Online embeddings** for higher semantic accuracy
  - **Offline TF-IDF fallback** for offline reliability and speed
- Optional **LLM-generated preference sentence** blended into the user vector to improve personalization
- Built for **speed and stability** when external services are unavailable

LLM Preference Sentence (OpenAI → Vector Boost)

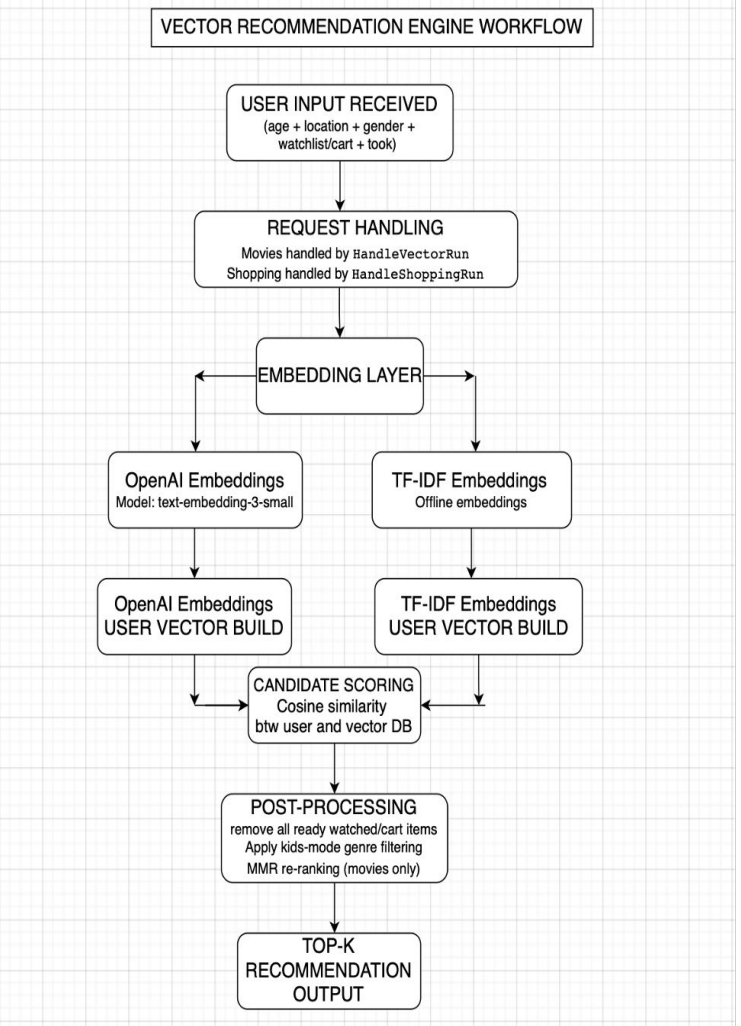
- Used to **improve personalization** in the vector recommendation engine
- OpenAI generates **one short preference sentence** from user profile + history
- Separate prompts for:
  - **Movies** (genres, tone, content level)
  - **Shopping** (categories, style, budget)
- Sentence is **embedded along with** user vector
- Fully **optional and safe**:
  - Runs only if OpenAI key is available
  - Automatically skipped or falls back if unavailable
- Keeps vector engine **fast, and reliable**, with accuracy boost

```
if e.catalogBase == "shopping" {
    systemMessage = "You are a shopping preference summarizer for a recommendation engine. " +
        "Given a user's profile and recent cart items, respond with exactly ONE short English sentence " +
        "that describes the kind of products they are most likely looking for. " +
        "Focus on product categories (electronics, fashion, home, gaming, etc.), " +
        "typical use cases (daily use, travel, work, fitness, etc.), and preferences " +
        "(budget-friendly, premium, brand-conscious, minimalist, colorful, etc.). " +
        "Do not mention movies, TV shows, or media. Do not address the user directly " +
        "and do not add explanations or bullet points."

    userContent = buildShoppingPreferenceSummaryUserContent(in)
} else {
    systemMessage = "You are a movie preference summarizer for a recommendation engine. " +
        "Given a user's profile and watch history, respond with exactly ONE short English sentence " +
        "that describes the kind of movies they are most likely looking for. " +
        "Focus on genres (for example: animation, kids, family, thriller, romance), tone " +
        "(light-hearted, dark, emotional, etc.), and content level (kid-friendly, avoid violence, etc.). " +
        "Do not address the user directly and do not add explanations or bullet points."
```

End-to-End Vector Recommendation Workflow

- User submits profile, preferences, and watchlist/cart
- Backend validates and normalizes input
- Request routed to movie/shopping vector pipeline
- User and catalog items embedded
- User profile vector construction
- Candidate items scored using cosine similarity
- Deterministic filtering and re-ranking applied
- Top-K recommendations returned to the client

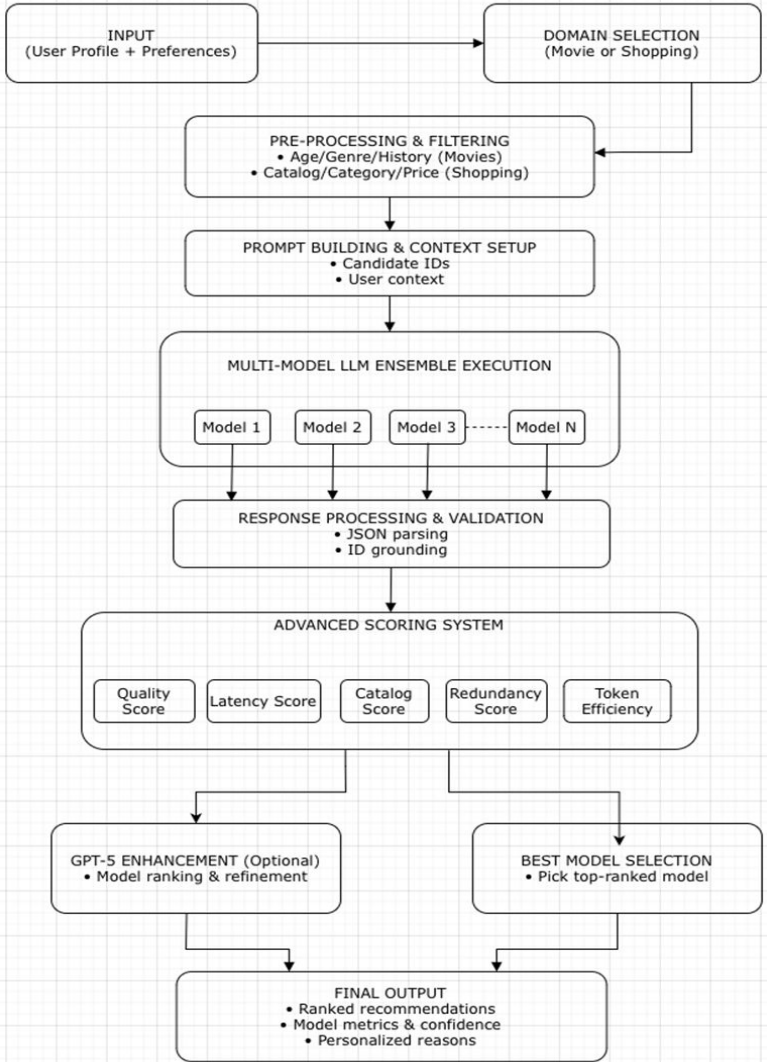


LLM Recommendation Engine Overview

- Built an AI-powered recommendation system for **movies and shopping**
- Designed as a **backend-first, production-style system**
- Uses **multiple LLMs** to generate and compare recommendations
- Focused on **reliability, consistency, and evaluation**
- Supports real user behavior like watch history and shopping carts

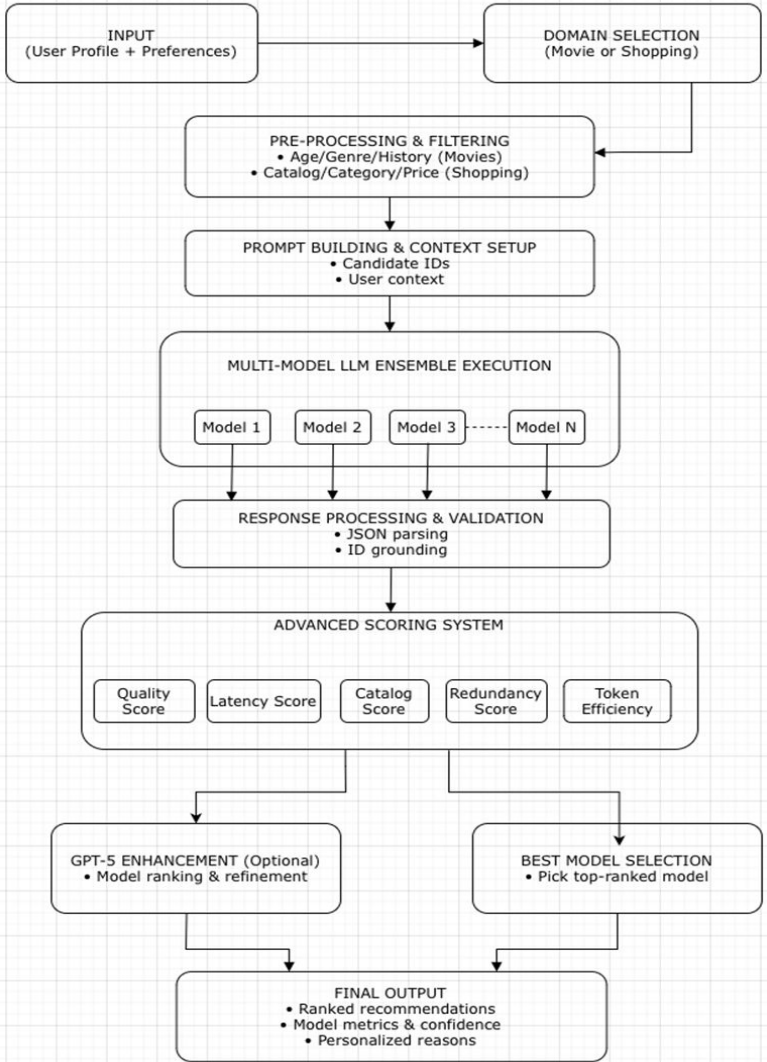
End-to-End Recommendation Workflow

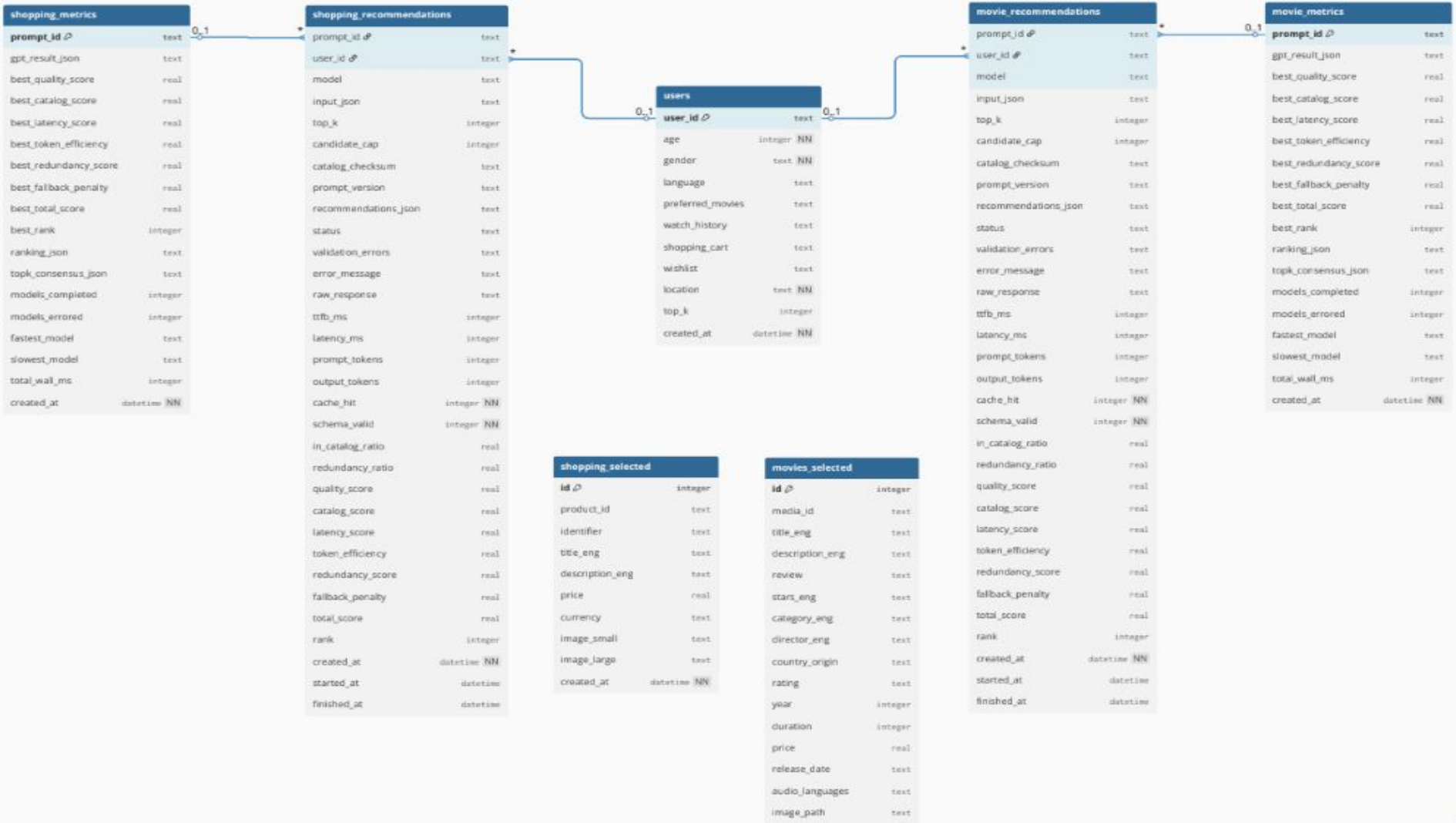
- User submits profile, preferences, and history
- Backend validates and normalizes input
- Candidate items fetched from catalog
- Candidates pre-filtered before LLM execution
- Prompt constructed with strict rules
- Multiple LLMs run in parallel
- Results validated, stored, and returned



LLM Execution & Result Handling

- Deterministic preprocessing for repeatable runs
- Parallel model execution with per-model isolation
- Timeout and failure handling per LLM
- Schema validation and grounding against catalog
- Centralized persistence of outputs and metrics




shopping_metrics
prompt_id text
gpt_result_json text
best_quality_score real
best_catalog_score real
best_latency_score real
best_token_efficiency real
best_redundancy_score real
best_fallback_penalty real
best_total_score real
best_rank integer
ranking_json text
topk_consensus_json text
models_completed integer
models_errored integer
fastest_model text
slowest_model text
total_wall_ms integer
created_at datetime NN
0..1
*
shopping_recommendations
prompt_id text
user_id text
model text
input_json text
top_k integer
candidate_cap integer
catalog_checksum text
prompt_version text
recommendations_json text
status text
validation_errors text
error_message text
raw_response text
ttfb_ms integer
latency_ms integer
prompt_tokens integer
output_tokens integer
cache_hit integer NN
schema_valid integer NN
in_catalog_ratio real
redundancy_ratio real
quality_score real
catalog_score real
latency_score real
token_efficiency real
redundancy_score real
fallback_penalty real
total_score real
rank integer
created_at datetime NN
started_at datetime
finished_at datetime
*
0..1
users
user_id text
age integer NN
gender text NN
language text
preferred_movies text
watch_history text
shopping_cart text
wishlist text
location text NN
top_k integer
created_at datetime NN
0..1
*
movie_recommendations
prompt_id text
user_id text
model text
input_json text
top_k integer
candidate_cap integer
catalog_checksum text
prompt_version text
recommendations_json text
status text
validation_errors text
error_message text
raw_response text
ttfb_ms integer
latency_ms integer
prompt_tokens integer
output_tokens integer
cache_hit integer NN
schema_valid integer NN
in_catalog_ratio real
redundancy_ratio real
quality_score real
catalog_score real
latency_score real
token_efficiency real
redundancy_score real
fallback_penalty real
total_score real
rank integer
created_at datetime NN
started_at datetime
finished_at datetime
*
0..1
movie_metrics
prompt_id text
gpt_result_json text
best_quality_score real
best_catalog_score real
best_latency_score real
best_token_efficiency real
best_redundancy_score real
best_fallback_penalty real
best_total_score real
best_rank integer
ranking_json text
topk_consensus_json text
models_completed integer
models_errored integer
fastest_model text
slowest_model text
total_wall_ms integer
created_at datetime NN
shopping_selected
id integer
product_id text
identifier text
title_eng text
description_eng text
price real
currency text
image_small text
image_large text
created_at datetime NN
movies_selected
id integer
media_id text
title_eng text
description_eng text
review text
stars_eng text
category_eng text
director_eng text
country_origin text
rating text
year integer
duration integer
price real
release_date text
audio_languages text
image_path text

Why Schema-Driven Design Matters

- Client required a **simple, lightweight database** with minimal operational overhead
- Embedded design supports **offline and edge environments** (e.g., inflight systems)
- Database acts as both **persistent storage and caching layer**
- **Users table** stores profile, preferences, history, cart, and wishlist
- **Catalog tables** provide a trusted, curated source for valid recommendations
- **Recommendation tables** store per-model outputs, raw responses, and metrics
- **Metrics tables** enable scoring, ranking, and model comparison
- Prompt-based structure enables **auditability, reproducibility, and caching**

Candidate Pre-Filtering

LLMs reason over **filtered candidates**, not full catalog

Reduces candidate pool from 290+ to ~50–100 items

Filtering criteria:

- Age and safety constraints
- Watch history exclusion
- Genre/category relevance
- Metadata quality checks

Improves accuracy, reduces hallucinations, lowers cost

Prompt Engineering: Purpose & Structure

- LLM operates as a **backend execution component**
- Prompt explicitly defines **system role and output constraints**
- Strict **JSON schema** enforced for all model responses
- User context included with **explicit behavioral interpretation rules**
- Recommendation choices limited to **pre-approved catalog IDs**
- Mandatory personalization, ordering, and diversity rules applied
- Uniform prompt structure enables **consistent validation and comparison**

Model Scoring Framework

- Objective: automated evaluation of multiple LLMs
- Designed for production reliability, not intuition
- Same inputs → fair comparison across models

Multi-Dimensional Scoring

Score	Weight	Definition
Quality Score	35%	Measures how relevant, personalized, diverse, and novel the recommendations are for the user.
Genre Match Score	40% of Quality	Measures alignment between user watch history and recommended content using cosine similarity.
Reasoning Score	25% of Quality	Evaluates explanation quality; penalizes generic or fallback reasoning.
Diversity Score	20% of Quality	Rewards variety by measuring unique genres across recommendations.
Novelty Score	15% of Quality	Encourages discovery by rewarding recommendations not seen in watch history.
Latency Score	35%	Rewards faster models; scores normalized relative to the fastest model per run.
Catalog Score	10%	Ensures all recommended items exist in the approved catalog; penalizes hallucinated items.
Token Efficiency Score	10%	Rewards models that generate concise outputs close to the optimal token range.
Redundancy Score	10%	Penalizes duplicate recommendations and excessive repetition.

AUTOMATED SCORING & GPT ADJUDICATION

- All models run in parallel
- Scores computed automatically
- **Primary selection: score-based ranking**
- **Optional GPT-5 adjudication for semantic review**

GPT-5 ADJUDICATION

Score	Definition
Personalization Score	Measures how well recommendations align with user demographics, preferences, and watch history.
Content Quality Score	Evaluates intrinsic recommendation quality, including relevance, diversity, and novelty.
Technical Excellence Score	Assesses correctness of output, reasoning clarity, format compliance, and catalog grounding.
Business Value Score	Estimates practical value based on user satisfaction and engagement potential.
Overall GPT Score	Weighted aggregate of all GPT evaluation dimensions, used for final model ranking.
Confidence Score	Indicates GPT's confidence in its evaluation and ranking decision (0.0–1.0).

FINAL MODEL SELECTION

- **Primary:** phi3:mini
- **Fallback:** qwen2.5:3b-instruct
- Best speed-to-quality tradeoff
- Most consistent latency under SLA

Evaluated across 10 personas

TEAM ACCOMPLISHMENTS

- Built a **production-grade, multi-model LLM evaluation system**
- Designed a **multi-dimensional scoring framework** balancing quality, speed, and reliability
- Implemented **fully automated best-model selection** with explainable metrics
- Added **optional GPT-5 adjudication** without making it a single point of failure
- Delivered an **offline-capable, auditable, and deployment-ready** backend
- Implemented vector search with very accurate results.

FRONTEND

- **Ideanova Recommendation Hub**: AI-driven platform for personalized movie & shopping recommendations
- **Core Functionality**: Profile-based queries with multi-model AI (LLM + Vector search)
- **User Flow**: Set profile → Choose mode (Movies/Shopping) → View & compare real-time outputs
- **Target**: Get accurate and user preferred recommendatiosn

Key Features and Functions

- **Personalization Engine**: Tailored recs using age, gender, watch/cart history
- **Multi-Model Comparison**: Run & rank LLMs alongside Vector recs
- **History Tracking**: Real-time watch/cart updates with progress sliders
- **Leaderboard**: Previous runs with user inputs and outputs
- **Export CSV**
- **Stop Run-** Stops the current LLM run

EXPORT CSV

1	USER PROFILE													
2	Run ID	16f73a2e-633f-4bbd-87eb-7bff9d96817e												
3	User ID	9a71c266-8891-4310-8cc2-cb83e1916f72												
4	Age	22												
5	Gender	female												
6	Location	United States												
7	Top K	10												
8	Preferred Movies													
9	Started At	2025-12-17 10:51:10.618177 +0000 UTC												
10	Finished At	2025-12-17 10:55:00.724637 +0000 UTC												
11	Total Duration (ms)	230175												
12	Best Model	qwen2.5:1.5b-instruct												
13	Models Used	llama3.2:1b,llama3:latest,phi3:mini,qwen2.5-coder:1.5b,qwen2.5:1.5b-instruct,qwen2.5:3b-instruct,tinyllama:1.1b												
14														
15	TOP CONSENSUS RE	Support Count												
16	T-hairspray	7												
17	T-those_who_wish_n	7												
18	T-for_love_life_no_or	5												
19	int_02045	5												
20	T-barbie	4												
21	T-elvis	4												
22	int_018	3												
23	T-fantastic_beasts_a	2												
24	T-42	2												
25	int_02046	2												
26														
27	MODEL RESULTS													
28	Model	Rank	Total Score	Quality Score	Catalog Scor	Latency Scor	Token Efficie	Redundancy	Fallback Pen	Latency (ms)	Prompt Toker	Output Toker	In Catalog Ra	Redundancy	Recommendations	
29	qwen2.5:1.5b-instruc	1	100.37	104.5	100	100	88	100	20	19205	2898	240	10	0	1. int_02045: Matches profile | 2. T-barbie: Matches profile | 3. int_018: Matche	
30	qwen2.5-coder:1.5b	2	96.35	93.5	100	99.5	88	100	60	19301	2898	240	10	0	1. T-42: Matches profile | 2. T-rebel_without_a_cause: Matches profile | 3. T-aqu	
31	phi3:mini	3	82.45	117.5	100	35.78	88	100	60	53677	3532	240	10	0	1. int_02068: Matches profile | 2. T-the_lord_of_the_rings_the_two_towers: Mat	
32	tinyllama:1.1b	4	75.65	92.5	100	44.22	78	100	100	43430	2048	190	10	0	1. T-those_who_wish_me_dead: fallback pick | 2. T-hairspray: fallback pick | 3. T	
33	qwen2.5:3b-instruct	5	72.85	102.5	100	23.36	88	100	20	82219	2898	240	10	0	1. T-hairspray: Matches profile | 2. T-creed: Matches profile | 3. int_02045: Matc	
34	llama3.2:1b	6	63.19	74.5	100	23.76	88	100	20	80837	2770	240	10	0	1. T-300_rise_of_an_empire: Matches profile | 2. T-the_hobbit_the_battle_of_the	
35	llama3:latest	7	58.67	77	100	8.35	88	100	40	230081	2755	240	10	0	1. T-the_notebook: Matches profile | 2. T-fantastic_beasts_and_where_to_find_t	
36																

Highlights

- React for modular and Framer Motion for smooth UI (hover effects,model comparison)
- LLM and Vector-based recommendation
- Reusable UI elements and shared logic
- Metrics for each and every model
- System supports running up to 7 AI models per user
- Execution completes in ~3–7 minutes

CHALLENGES FACED

Different models returned data in different formats, so we handled multiple field names and added safe defaults in the UI.

Some recommendations had missing or broken images, so we added fallback images to prevent the page from breaking.

Some models returned empty or failed results, so we showed clear “no recommendations” messages instead of crashing.

INDIVIDUAL CONTRIBUTIONS

- **Honey Reddy Nagireddy:** Backend architecture, database design, LLM & shopping recommendation integration, caching and optimization, API development, prompt engineering, and system documentation
- **Ssai Sanjanna Ganji:** Backend architecture, LLM scoring and ranking framework, automated model selection, GPT-5 adjudication, system integration, documentation
- **Preethi Reddy Nagireddy**: Responsible for building the front end of the project, implementing all required features from user interaction and personalization to AI recommendations, model comparison, and result analysis, documentation
- **Siddhartha Gudipudi:** Initial research on Ollama, python scripts to run LLM's one after other via Ollama, Vector architecture, Vector implementation, documentation

DEMO