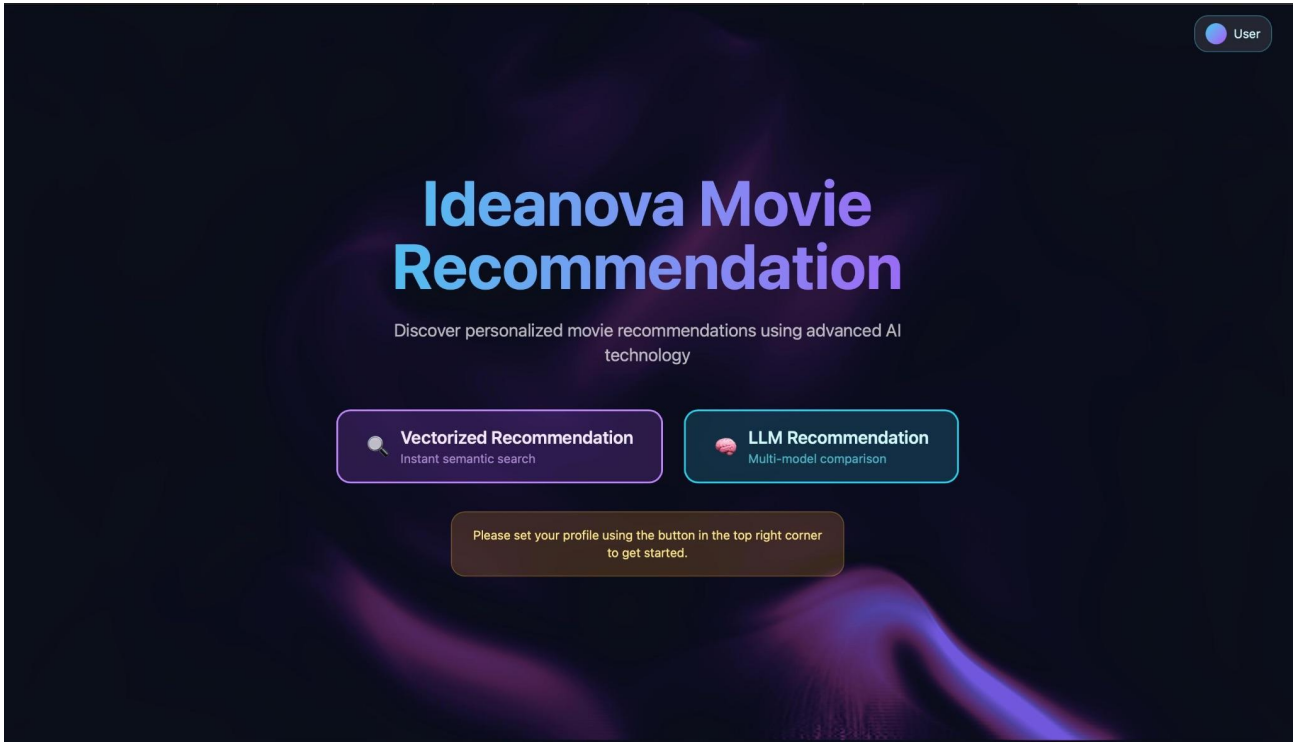
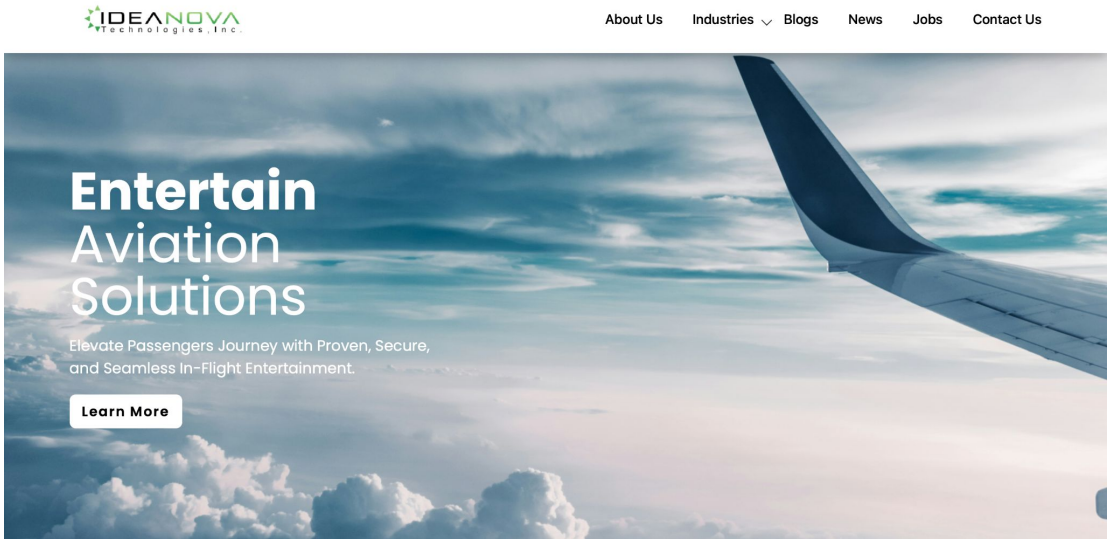
AI Recommendation Engine



–**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

WHO - The Client

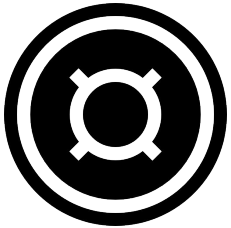


IdeaNova Technologies Inc.

Our client is a media streaming platform aiming to offer personalized, engaging content recommendations, ensuring both speed and explainability in suggestions.

- Develops media portals for inflight entertainment systems
- Seat displays in airplanes
- Users: Passengers accessing movies, shopping, and other services

WHY - The Problem



Generic recommendations
– All users see similar content



Minimal personalization
– Doesn't consider viewing/shopping history



Poor user experience
– Irrelevant suggestions

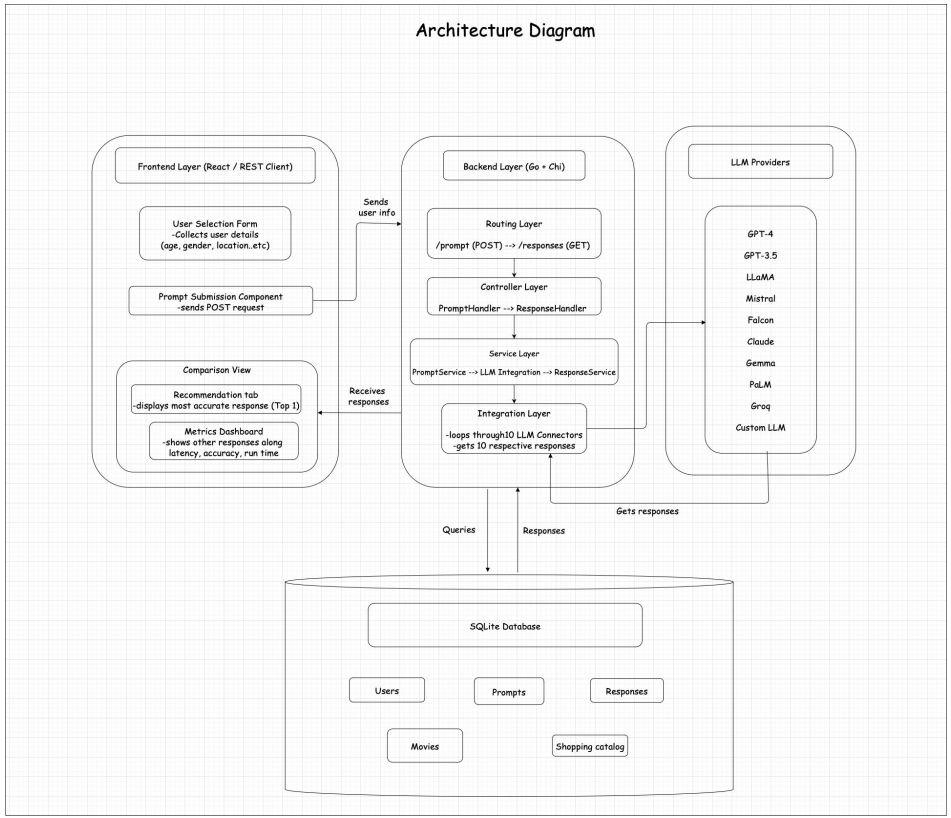
HOW - Our Solution

Our solution uses a **recommendation system**: an LLM Engine for reasoning and explainable results, and also a secondary Vector Engine for fast similarity-based retrieval. Both engines expose clean APIs and track key metrics like latency, accuracy, and redundancy.

AI-Powered Recommendation Backend

- Tracks user viewing and shopping history
- Leverages Large Language Models (LLMs) to generate personalized recommendations
- Integrates with IdeaNova's existing frontend
- Cross-catalog recommendations (movies ↔ products)

System Architecture and Components



- **Frontend Layer** (React/REST Client)
- **Backend Layer** (Go + Chi framework)
- **LLM Integration Layer** (Multiple providers)
- **Database** (SQLite)

Design Challenges

Challenge 1: LLM Provider Abstraction

- Must support multiple LLM providers (OpenAI, Anthropic, local models)
- Different APIs, response formats, authentication methods
- ***Solution:*** Unified integration layer with connector pattern

Challenge 2: Recommendation Quality & Evaluation

- How do we measure if recommendations are "good"?
- Need empirical metrics, not subjective opinions
- ***Solution:*** Configurable evaluation framework with automated testing

Challenge 3: Offline Environment Support

- Airplanes have limited/no internet connectivity
- Must support locally hosted models
- ***Solution:*** Architecture supports both online and local models

Tech & Tools Used

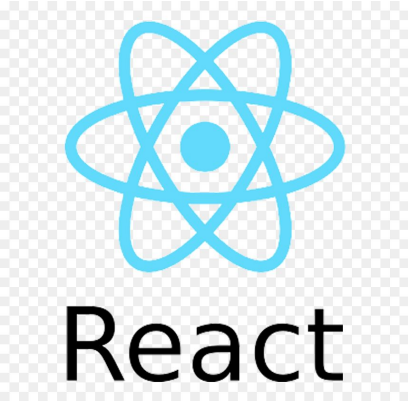








Tech & Tools Used







Database Schema

Users – Stores user profile info *(user_id, age, gender, language, location etc)*. **PK:** user_id.

Prompts – One row **per (prompt_id, model)** with the input, per-model results, and per-model metrics.

- **PK:** (prompt_id, model) (composite).
- **FK:** user_id → users.user_id

Metrics – **One summary row per prompt_id** (aggregated run metrics).

- **PK:** prompt_id.
- **No FK declared** to prompts (logical 1:1 via shared prompt_id).

Movies Catalog (movies_selected) – Curated movie metadata (title, year, rating, image_path, etc.).

- **PK:** id (autoincrement).
- **No FKs** (catalog is standalone)

Backend API's

- **Endpoint**: /api/llm/recommendations/run
- **Input (POST body)**: User profile (age, gender, location, preferences, topK)
- **Process**:
  - Runs multiple LLMs (Qwen, Gemma, Llama etc)
  - Each model generates recommendations
  - Results grounded against catalog
  - Metrics tracked (latency, schema validity, in-catalog ratio, redundancy)
- **Output (JSON)**:
  - Run ID, input profile
  - Per-model recommendations with metrics
  - Consensus / adjudicated final list

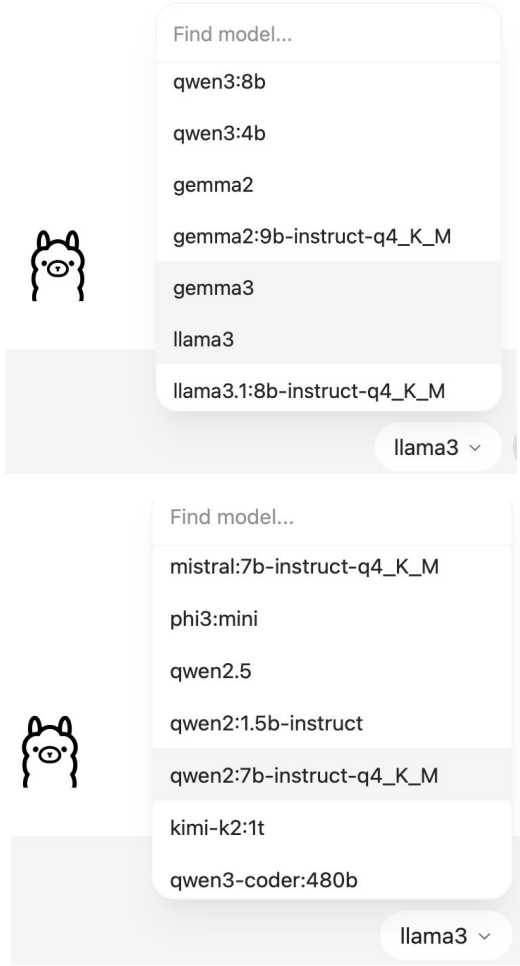
Backend API's

Endpoint: /api/v1/recommendations/vector

- **Input (POST body)**:
  - User profile (age, gender, location, preferences, topK)
- **Process**
  - Uses embeddings & vector similarity search
  - Finds closest matches from movie catalog
- **Output (JSON)**: List of recommended movies.

Research on top LLMs

- Evaluated open-source LLMs
- Personas: age, gender, location, preferences
- Run one LLM after other
- Record the outputs



How we are running the scripts

- Automated evaluation.
- Runs each model against catalog with persona prompts
- Saves raw outputs, parses JSON, logs metrics
- Leaderboard + full JSON report

How we are validating the models

- Each model receives a **total score** and is ranked against others.
- The total score is based on:

    a. **Quality**
    b. **Latency**
    c. **Catalog info**
    d. **Tokens**
    e. **Redundancy**

model	quality_score	catalog_...	latency_score	token_ef...	redunda...	fallback...	total_score	rank
phi3:mini	55.00000000000001	100	100	88	100	30	85.3	1
qwen2.5:3b-instruct	60	100	54.318533244974596	88	100	20	73.09555997349239	2
llama3:latest	100	100	18.221563542052614	69.4	100	0	72.40646906261578	3
tinyllama:1.1b	19.999999999999996	100	78.44415075422685	45.4	100	100	64.07324522626806	4
qwen2.5:1.5b-instruct	0	0	0	0	0	0	0	5

Embedding and vectorizing

- We use **sentence-transformers/all-MiniLM-L6-v2** to convert movies into dense vector representations.
- Each movie is encoded using its title, synopsis, and genres to capture semantic meaning.
- These vectors are stored in a **FAISS HNSW index**, which allows for very fast similarity search at scale.
- A user's **persona (age, gender, location, preferences)** is also encoded into a vector, enabling us to compare it directly with the movie vectors.

How outputs are calculated through vectors

- Build a **candidate pool** using FAISS nearest-neighbor search on movie vectors.
- Score candidates by combining **similarity, popularity, freshness, and segment fit**.
- Use a **weighted blend** (0.60*Similarity + 0.25*Popularity + 0.10*SegmentFit + 0.05*Freshness) to balance relevance and diversity in results.
- Optionally we can apply a **CrossEncoder/MiniLM reranker** for more precise ranking.

```
(.venv) siddu123@MacBook-Air VecIndex % ./run_pipeline.sh
Step 1: Retrieve candidates with FAISS
[serve] index d=384, ntotal=352
Loading embedder from: /Users/siddu123/Desktop/SeniorDesign/sd06_personalization_1/VecIndex/models/all-MiniLM-L6-v2
{"latency_ms": 124, "titles": ["The Underdoggs", "Hard Miles", "Those Who Wish Me Dead", "The Alto Knights", "Twisters", "Ricky Stanicky", "Looper", "Keanu", "Rain Man", "Vlad and Lousie"]}
[serve] wrote 10 candidates -> candidates.json
Step 2: Rerank candidates
Loading CrossEncoder from ./models/crossencoder on cpu
Reranked results saved to reranked.json
Final reranked recommendations saved to reranked.json
```

CURRENT PROGRESS

Finalizing vector engine implementation – Working on completing the similarity-based recommendation system for fast content matching

Integrating LLM engine with Ollama & OpenAI – Connecting multiple language model providers for intelligent recommendations

Building database schema with auto-ensure – Developing robust data structure that automatically maintains data integrity

Connecting frontend to backend APIs – In progress of linking user interface with server-side services

Enhancing metrics tracking efficiency – Improving per-run performance monitoring for better insights

Exploring open-source GPT APIs – Researching and testing additional online language models for integration

System-wide optimization – Refining overall performance across all components

Expanding model coverage – Scaling from 5 to 10 different models to improve recommendation quality

Frontend optimization in progress – Improving user interface speed and responsiveness

CHALLENGES

Invalid JSON from LLMs – Language models occasionally return malformed responses; addressed through strict prompt engineering

Model latency gaps – Different LLMs have varying response times, creating inconsistent user experience

Duplicate recommendations – System sometimes suggests the same content multiple times in results

Separate asset serving – Movie posters and images require independent hosting infrastructure

Metric evaluation criteria – Determining the best benchmarks to measure recommendation effectiveness

Model selection complexity – Identifying which LLMs deliver optimal balance of speed and accuracy

FUTURE WORK

Research advanced models – Exploring additional online and open-source language models for better performance

Balance vector and LLM efficiency – Since LLMs have longer response times, optimizing both engines to work seamlessly together

Extend to shopping recommendations – Applying combined vector and LLM approach to e-commerce product suggestions

Run on **Raspberry Pi 5 with Docke**r plus basic monitoring and auto-updates**.**

User feedback loop – Enabling ratings and preferences to continuously improve recommendations

Prompt optimization – Refining LLM instructions for more consistent and reliable outputs

Conclusions



The project addresses media recommendation challenges with a **fast, explainable dual-engine system**, promising a scalable and personalized user experience enhanced by ongoing research and cloud adaptation.