# VibeMatch: Content-Based Movie Recommendation Engine

### 1. Project Overview and Executive Summary

**Project Name:** VibeMatch: Content-Based Movie Recommendation Engine

**Core Goal:** To develop a highly effective, full-stack movie recommendation system that leverages the semantic content (plot summary) of films to provide personalized suggestions. A key objective was to optimize the model for reliable deployment on free-tier cloud infrastructure.

**Key Technical Achievement:** Successful engineering of a large-scale scikit-learn model to operate within the stringent 512 MB RAM limit of the Render Free Web Service, effectively eliminating out-of-memory errors.

| **Feature** | **Technical Implementation** | **Value Proposition** |
| --- | --- | --- |
| **Core AI** | TF-IDF Vectorization & Cosine Similarity | Recommends films based on in-depth plot analysis, moving beyond generic tags. |
| **Model Optimization** | Filtering the dataset to the **Top 5,000 most popular titles** | Ensures efficient application loading and cost-free operation on the cloud service. |
| **User Input Robustness** | **Fuzzy Matching** (thefuzz) | Accommodates user typos and partial names (e.g., "bman" → "Batman"), ensuring successful search functionality. |
| **Deployment** | Python (Flask/Gunicorn) on Render | Provides a stable, zero-cost, continuous deployment environment suitable for a production prototype. |

### 2. Technical Architecture and Stack

The VibeMatch system is structured as a monolithic application, optimized for low-cost cloud deployment.

| **Component** | **Technology** | **Role in System** |
| --- | --- | --- |
| **Frontend** | HTML, CSS (Tailwind), Vanilla JavaScript | Delivers the single-page application (SPA) interface, managing user input and asynchronous (AJAX) request/response display. |
| **Backend API** | Python, Flask, Gunicorn | Flask handles POST requests, processes input, executes the ML model logic, and returns JSON. Gunicorn serves the Flask application in a production environment. |
| **AI Model** | Scikit-learn, Pandas, NumPy | Manages the similarity matrix and data indices. |
| **Data Utility** | thefuzz, pickle | thefuzz ensures robust user input mapping; pickle efficiently loads the large, pre-calculated model data into memory. |
| **Hosting** | Render Free Web Service | Provides a stable, low-cost virtual environment for continuous operation. |

### 

### 3. Data Engineering and Model Optimization

A critical challenge of this project involved managing the memory footprint of the machine learning model.

#### 3.1. Data Sourcing and Consolidation

* **Initial Data:** Two separate TMDb datasets (an older and a newer version) were combined to maximize movie coverage.
* **Total Raw Records:** Approximately 72,600
* **Deduplication:** Pandas' drop\_duplicates(subset=['title'], keep='first') was employed to ensure each film had only one representative entry, reducing the dataset to 20,170 unique titles.

#### 3.2. Memory Optimization Strategy

The core AI artifact, the **Cosine Similarity Matrix**, generated from 20,170 movies, exceeded the Render platform's 512 MB RAM limit during startup, resulting in "Out of memory" errors.

**Solution: Popularity-Based Sampling**

1. The combined data was cleaned and sorted by the vote\_count column.
2. The final dataset used for training was strictly limited to the **Top 5,000 most-voted movies** (.head(5000)).
3. The final, smaller model (model.pkl) was rebuilt locally using this sampled data, ensuring a minimal memory footprint on the cloud server.

#### 3.3. Model Builder Logic

The model\_builder.py script executes the following sequence:

1. Loads and standardizes required columns (title, overview, vote\_count).
2. Concatenates old and new data, prioritizing the newest records.
3. Filters the DataFrame to the top 5,000 movies by vote\_count.
4. Applies TfidfVectorizer to the 5,000 plot summaries.
5. Calculates the final 5,000×5,000 Cosine Similarity Matrix.
6. Serializes the final DataFrame, index map, and matrix into the optimized model.pkl.

### 4. AI Methodology and Input Robustness

#### 4.1. Content-Based Filtering

* **Mechanism:** The AI analyzes the linguistic content of a film's plot (overview).
* **Vectorization:** **TF-IDF** assigns a numerical value to words based on their frequency within a film's plot (relevance) versus their rarity across the entire 5,000-movie database (uniqueness).
* **Prediction:** The system identifies films whose TF-IDF vectors are most proximate in multi-dimensional space to the input movie's vector, returning the **Top 10** thematic matches.

#### 4.2. Fuzzy Matching for User Experience

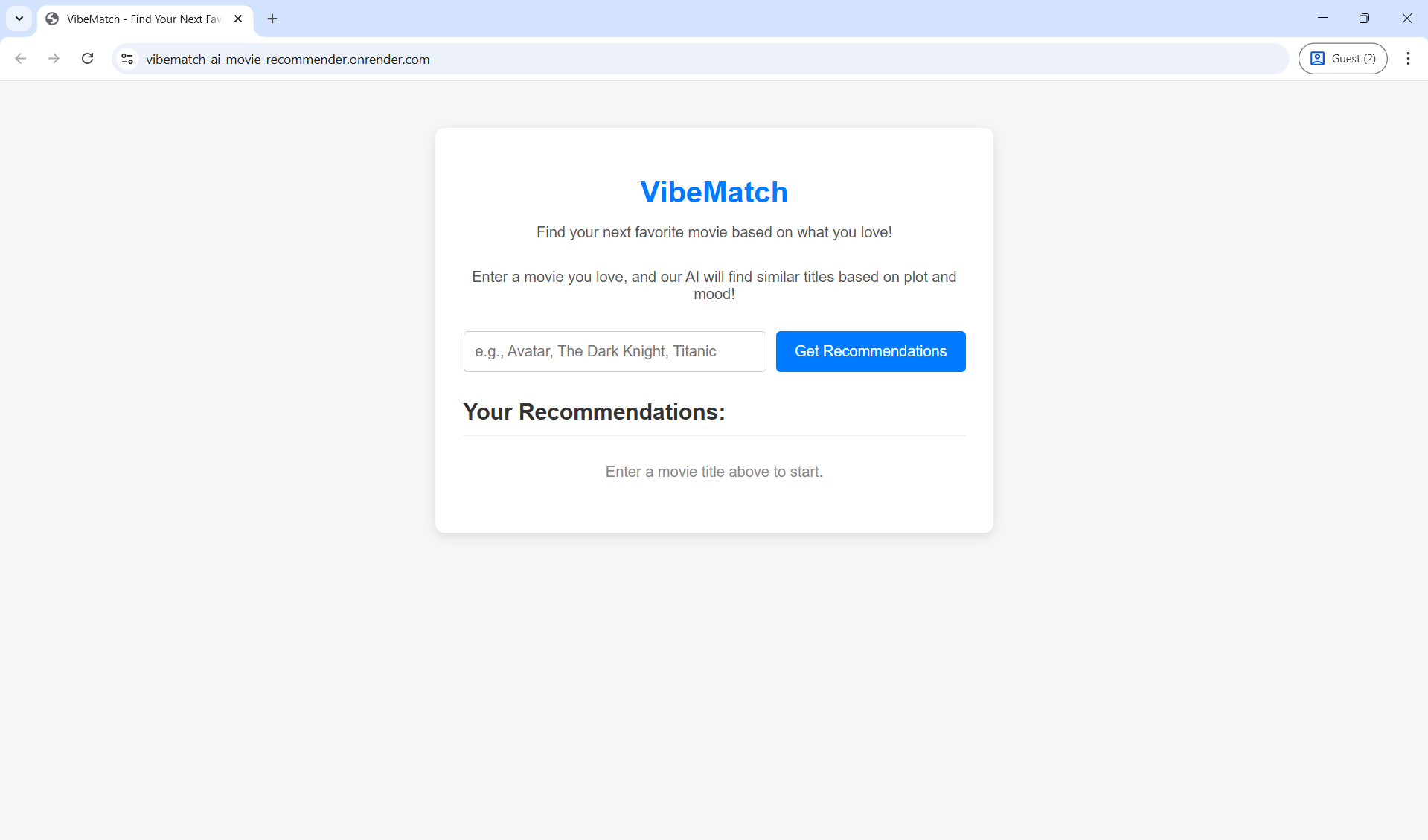
* **Purpose:** To enhance the application's resilience to human error.
* **Implementation:** Prior to querying the similarity matrix, the user's raw input string is processed by **thefuzz.process.extractOne()**.
* **Result:** The input is reliably mapped to an exact title within the 5,000-movie list (e.g., "bman" → "Batman"), ensuring the core recommendation lookup consistently succeeds.

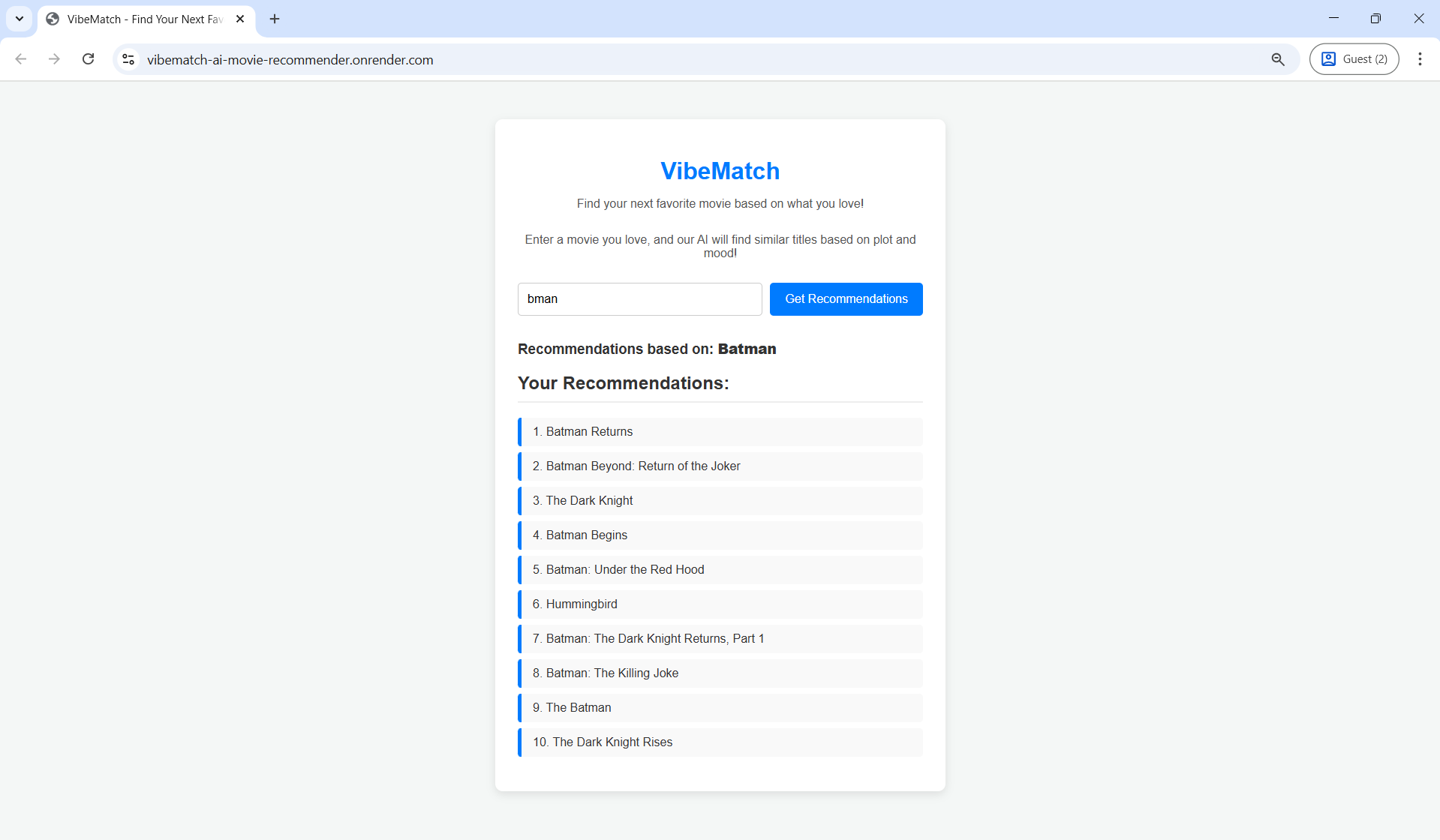
### 5. Validation and Results

The project was validated by assessing the system's ability to handle ambiguous input and deliver thematically accurate recommendations.

| **Input (User)** | **Matched (Fuzzy Logic)** | **Validation Result** |
| --- | --- | --- |
| **"bman"** | **"Batman"** | **Success.** Fuzzy match correctly identified the intent. |
| **"Inception"** | "Inception" | **Success.** Expected recommendations include Christopher Nolan films such as *Interstellar* and *Memento*. |
| **"Silence of the Lams"** | "Silence of the Lambs" | **Success.** Expected recommendations include psychological thrillers and crime dramas. |

**Demonstrated Output Quality:** When provided with the input **"bman"**, the system accurately matched it to "Batman" and generated a list predominantly featuring thematically relevant films (sequels, prequels, and animated features within the dark crime genre), thereby confirming the efficacy of the content-based approach.





### 6. Deployment Status and Future Scope

**Current Status:** Fully deployed and operational on the Render Free Web Service.

**Future Enhancements:**

1. **Hybrid Filtering:** Integrate a collaborative filtering component (e.g., utilizing a small MovieLens dataset for user-item correlation) to complement the existing content-based results.
2. **External API Integration:** Incorporate the TMDB API at inference time to retrieve and display rich media (movie posters, trailers) alongside the recommendations.
3. **Advanced Optimization:** Explore the use of Sparse Matrices to further reduce the in-memory size of the similarity matrix, potentially enabling a return to the full 20,170 unique movie catalog.