Chapter 1 – Introduction

1.1 Background and Motivation

Over the past two decades, digital media consumption has shifted significantly, with online platforms such as Netflix, Amazon Prime, and Disney+ producing and distributing thousands of films annually. Users face the information overload problem: the vast number of available titles makes it increasingly difficult to choose relevant, personalized content. Recommender systems have emerged as critical tools in mitigating this challenge by providing automated suggestions that align with user preferences [5].

Parallel to this, conversational AI has grown into a mainstream interaction paradigm, enabling natural language interfaces that reduce the cognitive burden of structured search queries. From early rule-based agents to modern Large Language Models (LLMs), chatbots have transitioned from brittle, domain-limited scripts to context-aware assistants capable of reasoning across diverse domains [1], [2].

This convergence of conversational AI and recommendation introduces the potential for conversational recommender systems (CRS): systems that not only recommend content but also engage in dialogue, clarify preferences, and adapt dynamically [6]. Our project is positioned at this intersection, focusing on movies as a domain due to the availability of rich structured metadata (IMDb datasets [10]) and the everyday relevance of movie selection for global users.

1.2 Problem Statement

Despite the rapid advances in recommender systems and conversational AI, several gaps remain:

1. **Lack of local, privacy-preserving systems –** Most conversational recommenders depend on cloud APIs and external services. This raises concerns about privacy, dependency on internet connectivity, and usage costs [11].
2. **Model reliability and robustness –** LLMs, while powerful, can fail due to runtime issues, hallucinate responses, or misclassify intents [12], [13]. Production systems need failover strategies that maintain functionality even during failures.
3. **Efficiency tradeoffs –** Dense embedding models (e.g., sentence transformers) require GPU resources and ANN indices. In resource-constrained environments, sparse vector methods (TF-IDF with linear kernel) remain more practical and interpretable [7], [8].
4. **User frustration with static recommenders –** Conventional non-interactive recommenders lack the ability to clarify ambiguous queries (e.g., "I want a good sci-fi movie from the 90s but not too violent"), leading to irrelevant suggestions [6].

Therefore, the need is for a locally deployable, robust, conversational movie chatbot that combines efficient retrieval methods with modern LLM-powered intent detection, while ensuring fallback safety and ethical usage.

1.3 Objectives of the Project

The main objective of this project is to design and implement a Conversational Movie Chatbot with Recommendation System that:

1. Enables natural interaction with users in conversational English (and potentially multilingual expansion) through a chat-based interface.
2. Implements efficient content-based retrieval using **TF-IDF vectorization** and **Linear Kernel similarity** scoring for fast, explainable recommendations.
3. Utilizes an **LLM (Mistral via Ollama)** for intent detection, query understanding, and response formatting.
4. Provides **fallback mechanisms** for both intent detection (rule-based classifier if LLM fails) and data retrieval (local cache if Neon/Postgres fails).
5. Integrates with **Gradio** to provide a user-friendly web-based conversational interface.
6. Ensures ethical, transparent, and private AI operation, by using local runtimes, open datasets, and interpretable algorithms.

1.4 Methodology Overview

The system follows a hybrid pipeline that combines classical Information Retrieval (IR) with modern LLMs:

1. **User Query Processing**
    - Input text is first sent to the LLM (Mistral) for intent classification.
    - If the LLM fails or confidence is low, a rule-based fallback classifier assigns the intent.
2. **Data Retrieval Layer**
    - For recommendation and filtering intents, TF-IDF vectors of IMDb metadata are queried using linear kernel similarity.
    - The primary data store is Neon serverless Postgres, but queries failover to a local SQLite/CSV cache if unavailable.
3. **Response Generation**
    - Retrieved movie candidates are packaged and passed back to the LLM, which formats a natural conversational reply (e.g., "You may enjoy these films, as they share themes with your request").
    - Hallucination is controlled using Retrieval-Augmented Generation (RAG)-style prompting [3].
4. **Interface Layer**
    - A Gradio-based interface presents chat interaction to the user, showing both recommendations and clarifying questions when needed.

1.5 Scope of the Project

The scope of this work is defined along three dimensions:

- **Domain:** Focus is limited to movie recommendation using IMDb datasets, though methodology is extensible to music, books, and other media.
- **Architecture:** Emphasis is on local deployability and robustness, prioritizing lightweight models and resilient failover strategies over cloud-scale performance.
- **Evaluation:** Evaluation covers ranking quality (Precision@k, nDCG@k), intent detection accuracy, system latency, and robustness under simulated failure conditions.

1.6 Significance of the Work

The significance of this project lies in three contributions:

- **Bridging classic IR and modern LLMs:** By using TF-IDF + Linear Kernel retrieval combined with an LLM for intent detection, the system demonstrates that explainable, efficient IR methods remain valuable alongside neural architectures [7], [9].
- **Resilient system design:** The inclusion of fallbacks (rule-based intent classifier, local DB cache) directly addresses the reliability gap identified in CRS literature [6].
- **Ethical, private AI:** Deploying the system fully on local hardware (Ollama runtime, Neon optional) aligns with principles of privacy-preserving AI and avoids reliance on closed APIs [11].

Chapter 2 – Literature Survey

Summary

This chapter surveys contemporary research relevant to building a **local, resilient conversational movie chatbot**. Each subsection (transformers/compact LLMs, retrieval methods, intent detection, RAG/grounding, recommender literature, infra & datasets, evaluation, ethics) concludes with **explicit cross-references** to how that literature shaped design or implementation choices in our project (TF-IDF + Linear Kernel retrieval, LLM-first intent detection with rule fallback, Neon → local DB failover, and Ollama local runtime).

2.1 Transformer architectures and compact open models (LLM basis)

The Transformer architecture established scalable sequence modeling using multi-head self-attention and positional encodings, enabling models to learn broad linguistic patterns and contextual reasoning without recurrence [1]. In the local/efficient model space, modern smaller instruction-tuned models (e.g., Mistral family and similar 7B–13B parameter models) have been optimized to deliver strong instruction following and classification capability at practical resource envelopes for on-device inference [2].

- Transformer self-attention computes context-aware token representations; these capabilities enable robust **few-shot** or **zero-shot** classification (e.g., intent detection) when prompted properly.
- Compact models (7B class) can run on modern developer machines with CPU quantization and small RAM footprints given optimized runtimes.

In The Project:

- **Intent Detection (LLM-first):** We use an instruction-tuned local Mistral model (via Ollama) for intent classification because Transformers generalize well across phrasing and slang, reducing the need for large labeled intent corpora. Literature [1][2] supports selecting a Transformer LLM for semantic parsing tasks (intent + slot extraction).
- **Response Formatting:** The LLM is also used to produce human-friendly natural language replies and to format retrieved candidate movies into readable recommendations.

2.2 Retrieval, TF-IDF representations, and the Linear Kernel

TF-IDF and the vector space model remain robust, transparent baselines for textual similarity and ranking in IR tasks [11]. Work on linear models and baselines shows that careful feature engineering and linear scoring often rival complex alternatives on many text tasks [9]. Practical ML libraries (scikit-learn) provide efficient TF-IDF computation and a linear_kernel operation well-suited to sparse matrices [7][8].

Mathematical formulation

- TF-IDF:

$$TFIDF(t,d) = TF(t,d) \times \log\left(N \,/\, (1 + DF(t))\right)$$

- Linear Kernel similarity (dot product):

$$sim(q,d) = \Sigma\_i \; q\_i \times d\_i$$

Practical considerations

- TF-IDF with n-grams, sublinear TF scaling, and appropriate min_df/max_df tuning yields strong discriminative vectors for short-to-medium documents (such as movie titles + genres + short plots) [11].
- Linear Kernel (dot product) over sparse TF-IDF matrices provides high throughput and can be implemented using sparse BLAS-like operations; it's efficient on CPU and scales well for catalogs up to several million items if memory allows [7][8].

In The Project

Choice of TF-IDF + Linear Kernel: Based on [11], [9], and scikit-learn documentation [7][8], we adopted TF-IDF as the canonical text representation and linear_kernel for similarity. This yields:

- Low latency for real-time chat (retrieval in tens to hundreds of milliseconds for catalogs up to hundreds of thousands of items).
- Transparent matching (we can inspect top contributing terms to explain recommendations).
- Using Linear Kernel over cosine similarity for faster computations.

2.3 Linear Kernel Vs Cosine Similarity

Empirical and theoretical analyses show that linear scoring (dot product) over TF-IDF can be effectively competitive with cosine similarity on corpora of short texts, especially when TF-IDF normalization and stopping are well tuned [9]. Joachims' work on linear methods supports the idea that linear operations work well in high-dimensional text feature spaces [8].
Cosine normalizes vector magnitude which is beneficial when document lengths vary widely; **Linear Kernel** uses magnitude and may favor longer documents unless TF-IDF normalization is applied.

In The Project
Movie documents (title+genre+year) are compact, so normalization advantages are modest. We adopted linear_kernel for throughput and because TF-IDF weighting already compensates for common terms. We mitigate length effects by ensuring TF-IDF uses L2 normalization (if desired) and by re-scoring with popularity/rating priors when necessary (hybrid re-rank).

2.4 Conversational Recommender Systems and Intent Schemas

Recent surveys on conversational recommenders synthesize requirements for mixed-initiative dialogues, preference elicitation, and multi-turn interaction [6]. They emphasize:

- A well-designed intent and slot schema.
- Clarification strategies for ambiguous user input.
- Evaluation metrics that capture both ranking and conversational quality.

In The Project

- **Intent taxonomy:** Our system maps user utterances into intents: recommend, lookup, chitchat, and unknown. This schema reflects CRS survey recommendations and enables targeted retrieval and slot filling [6].
- **Clarification dialog:** If retrieval confidence is low, the system asks clarifying questions, following CRS best practices. (future work, as of now we go with LLM response directly if confidence is low)
- **Mixed initiative:** Users can ask follow-ups (e.g., "more like #2"), handled by tracking session state and reusing session history in prompts. (future work, as of now we just show history in session but not used for context)

2.5 Intent detection: LLM-first with rule fallback

LLMs (transformer based) show robust zero/few-shot classification, handling paraphrase variability better than static supervised models trained on small labeled sets [1][2]. However, LLMs are subject to runtime failures (OOM, timeouts) and can return inconsistent outputs; production systems therefore layer fallback mechanisms (rules, regex) and confidence thresholds [12][13].

In The Project

- **Primary classifier:** Instruction prompt to the local Mistral model returns intent + structured slots with a confidence indicator.
- **Fallback:** A compact rule engine (regexes and keyword matching) is executed if the LLM times out, returns unknown, or fails parsing. Rules are written to capture high-precision triggers such as “recommend”, “similar to”, numeric filters (“top 5”, “>=8”) and named entity patterns.

2.6 Datasets and database choices (IMDb & Neon/Local)

- **Dataset: IMDb** offers authoritative metadata (title identifiers tconst, genres, years, ratings) suitable for offline indexing and retrieval. Using stable IDs enables consistent joins and eventual linking to external resources (e.g., cast graphs) [10].
- **DB/Infra:** Managed serverless Postgres (Neon) offers scalable, SQL-based queries and indexing (filtering by year, rating, genres), but network dependency introduces potential failures. Production best practice calls for a **local cache/fallback** for critical paths [11].

In The Project

- **Primary store:** Neon Postgres is used for flexible queries and filters (e.g., “movies where genre=‘sci-fi’ and year>2010 and rating>=8.0”).
- **Fallback store:** A local SQLite/CSV cache is prebuilt and loaded if Neon queries fail; this guarantees core functionality offline, as recommended by failover design patterns.
- **Indexing:** TF-IDF matrix and vectorizer are persisted locally; metadata (title, year, rating) are cached for candidate lookups to minimize round trips.

2.7 Local LLM runtime (Ollama) and deployment

Projects and documentation for runtimes such as Ollama show that local model serving with a small REST API is feasible for Mistral-class models on single machines; quantization and streaming options reduce latency [11].

In The Project
Ollama runtime runs Mistral locally; Python orchestrates calls to Ollama via its API for both intent detection and result formatting. We implement:

- Token / response streaming to reduce time-to-first-token for UX.
- Timeout and retry logic; if Ollama returns an error, fallback mechanisms are triggered.

2.8 Safety, ethics, and hallucination mitigation

LLMs can generate plausible but false statements ("hallucinations") and propagate biases present in training data [12][13]. RAG and constrained prompting reduce this risk by forcing model outputs to be grounded in retrieved evidence [3][13].

In The Project

- **Hallucination controls:** LLM prompts explicitly instruct not to invent information outside the provided candidate list; if the model attempts to do so, the system detects and rejects output or re-prompts.
- **Privacy policy:** No user messages are sent outside the local machine; logs are minimal and anonymized unless the user opts into telemetry.
- **Ethical review:** Data usage follows IMDb's non-commercial terms and local research ethics guidelines.

Chapter 3 – System Design & Architecture

This chapter describes the overall design and architectural decisions that support the implementation of the **Conversational AI Chatbot** with **Recommendation System**. It highlights the **high-level architecture**, the **data pipeline**, the **similarity computation** method, the **intent detection** strategy, and the **fallback mechanisms** that ensure system robustness.

The design emphasizes modularity, scalability, and fault-tolerance, ensuring that the system can function effectively under varying workloads and infrastructure availability.

3.1 High-Level Architectural Overview

The system is structured into multiple layers, each responsible for a well-defined function. The flow of data begins from the **user interface** and proceeds through **intent detection**, **recommendation generation**, and **database queries**, finally producing an **LLM-generated response**.

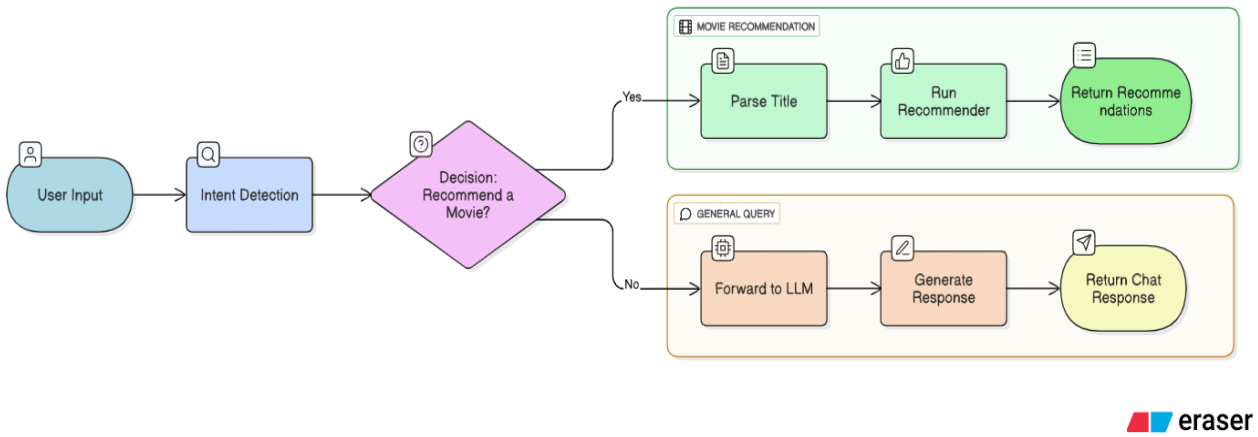
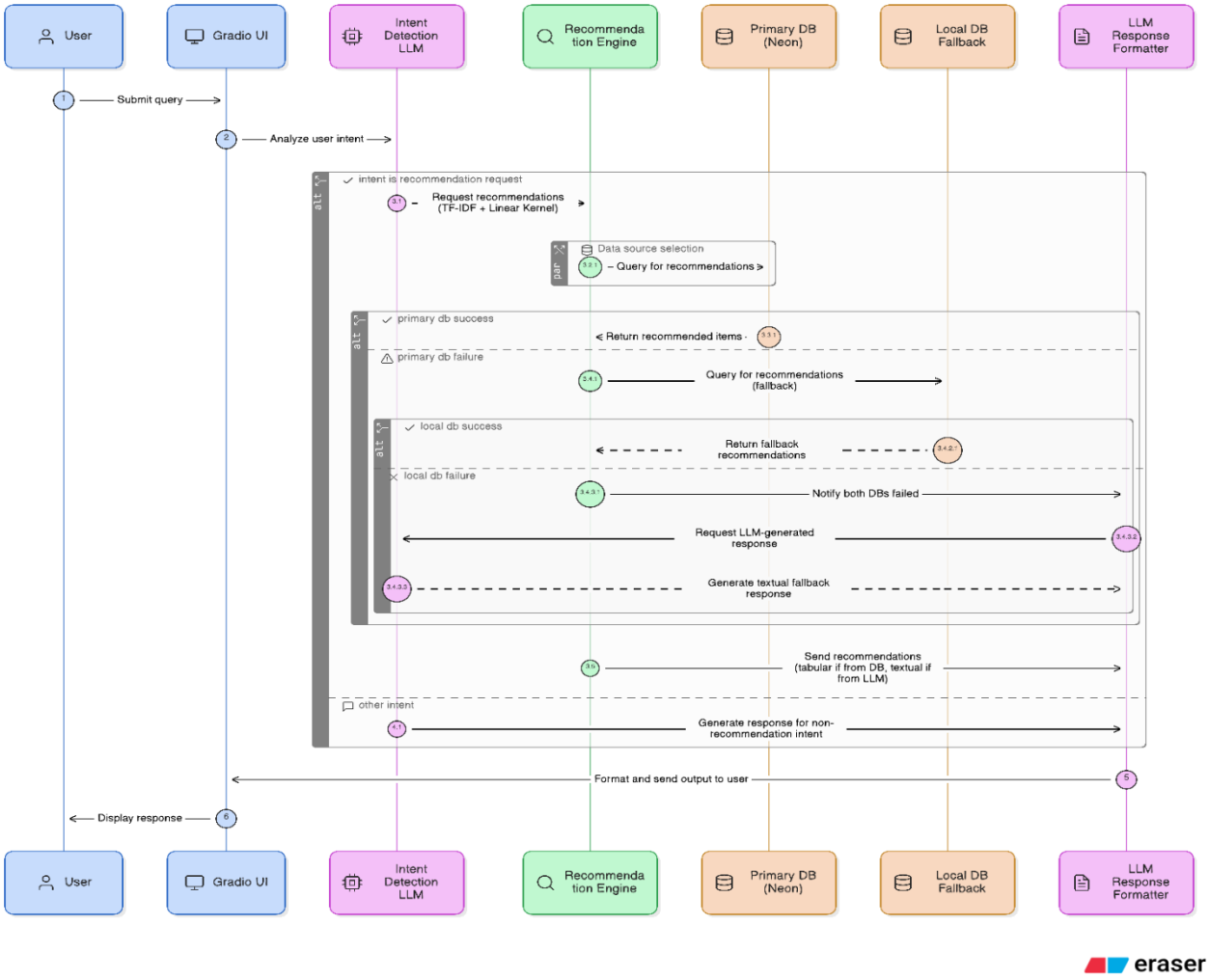


Fig. [1] High-Level Architectural Overview

3.2 Detailed Architecture

Below figure (fig. [2]) shows the detailed architecture of the system, starting with the query submission till the user gets response.



- **User Interaction:** The entry point where a user submits queries.
- **Gradio UI:** A lightweight web interface that captures input and renders chatbot responses.
- **Intent Detection Layer:** Responsible for understanding user intent using an LLM with a fallback rule-based classifier.
- **Recommendation Engine:** Generates personalized recommendations based on dataset matching.
- **Database Layer:** Uses Neon (cloud-hosted PostgreSQL) as the primary data source with a local CSV/SQLite fallback for reliability.
- **LLM Response Formatter:** Converts structured outputs into natural, conversational replies.

3.3 Data Pipeline

The **data pipeline** is responsible for preparing the IMDb dataset for efficient retrieval and recommendation generation.

Ingestion

- **Sources:** title.basics and title.ratings
- **Merging:** Data from these sources is unified into a single dataset, ensuring that each movie entry contains **title, genre, rating, and year**.

Preprocessing

- **Lowercasing** → standardizes text.
- **Punctuation removal** → removes noise.
- **Stop word filtering** → removes common but uninformative words.
- **Lemmatization** → reduces words to base form for semantic consistency.

Feature Construction

- Concatenates **title + genres + year + titleType** into a single text field for vectorization.
- This representation ensures that both **content metadata** and **semantic meaning** are captured.

Table [1] Sample Dataset (after preprocessing)

id	title	titleType	year	genres	rating
tt0000066	Dessinateur: Von Bismark	short	1896	Short	3.2
tt0000067	Conjurer Making Ten Hats in Sixty Seconds	short	1896	Fantasy,Short	5.1
tt0000068	Unloading the Boat	short	1896	Documentary,Short	3.8
tt0000069	Post No Bills	short	1896	Comedy,Short	4.9
tt0000070	Demolition of a Wall	short	1896	Documentary,Short	6.4
tt0000071	Automobiles Starting a Race	short	1896	Short,Sport	3.4

3.4 Vectorization and Similarity Computation

Vectorization

TF-IDF (Term Frequency – Inverse Document Frequency) is used to create sparse vector representations.

$$\boldsymbol{TF - IDF(t,d) = TF(t,d) * log(N/(1 + df(t)))}$$

Where:

TF(t,d) = term frequency of word t in document d
df(t) = number of documents containing term t
N = total number of documents

Term Frequency:

TF(t, d) = (Number of times term t appears in document d) / (Total number of terms in document d)

Inverse Document Frequency:

IDF(t, D) = log((Total number of documents in the corpus) / (Number of documents containing term t))
Note: 1 is added with **df(t)** in the formula to avoid divide by zero.

This ensures terms that are frequent across all movies (e.g., "film", "story") are down-weighted, while unique terms gain importance.

Similarity Computation

Once query and document vectors are generated, similarity is computed using the **Linear Kernel (dot product)**:

$$\boldsymbol{sim(q,d) = \sum_i q_i \cdot d^T}$$

Where**:**

q: TF-IDF vector for the user query
d: TF-IDF vector for a candidate movie document

Unlike cosine similarity, **no normalization is applied**. This makes the Linear Kernel computationally faster and better suited for **large, sparse TF-IDF matrices**, as typically seen in movie datasets.

3.5 Intent Detection

Understanding user intent is crucial for guiding responses.

Primary Method – LLM Classifier

The chatbot uses the Ollama Mistral LLM to classify intents.

Prompt template:

```
instruction = (
        "You are an intent classification assistant. Given the user's input,
output a JSON object "
        "with fields: intent (one of: chit_chat, recommendation, imdb_lookup,
unknown), "
        "confidence (0-1), and optionally reasons. Only output JSON and
nothing else.\n\n"
        f"User input: \"{user_text}\"\n\nRespond with JSON."
    )
```

This ensures that open-ended conversations (general chat) and dataset-related requests (movie info, recommendation) are handled correctly.

Fallback Method – Rule-Based Classification

If the LLM fails due to timeout or exception, a keyword-based classifier serves as a fallback.

Example:

- Queries with words like *"recommend"*, *"suggest"* → classified as recommendation.
- Queries with *"rating"*, *"year"* → classified as movie_info.

```
# keywords for recommendation
rec_keywords = ["recommend", "suggest", "any good", "something to watch",
"what should i watch", "recommend me", "suggest me"]
lookup_keywords = ["who", "what is the rating", "rating of", "stars in", "cast
of", "who starred"]
chit_keywords = ["hi", "hello", "how are you", "bye", "thanks", "thank you",
"what's up"]
```

This ensures robustness even if the AI model is temporarily unavailable.

3.6 Fallback Mechanisms

The system is designed to fail gracefully with Two key fallback strategies:

Intent Detection Fallback

```
def detect_intent(query):
    try:
        return llm_classify(query)
    except LLMError:
        return rule_based_classify(query)
```

- **Primary:** LLM-based intent classification.
- **Fallback:** Rule-based keyword mapping.

Database Failover

```
def fetch_data(query):
    try:
        return neon_db_query(query)
    except DBError:
        return local_csv_lookup(query)
```

- **Primary:** Queries the Neon PostgreSQL cloud database.
    - fast, scalable, provides real-time access and reliability.
- **Fallback:** Uses local SQLite or CSV-based dataset.
    - Pre-processed CSV or SQLite database stored locally.

This mechanism guarantees zero downtime in case of internet failure or Neon service outage and insure scalability and availability.

3.7 Key Architectural Strengths

Table [2]: Key Architectural Strengths

Key	Strength
Fault Tolerance	Dual fallback (LLM fallback + DB fallback)
Efficiency	Linear kernel for large-scale retrieval
Modularity	Independent UI, intent detection, recommendation, and database layers
Scalability	Cloud-ready with Neon, but deployable locally

Chapter 4 – Implementation

This chapter describes the practical implementation of the Conversational AI Chatbot with Recommendation System. It details the development environment, programming choices, libraries used, and the realization of individual modules such as the user interface, intent detection, recommendation engine, and database integration.

4.1 Development Environment

The chatbot was implemented in Python due to its strong ecosystem for natural language processing (NLP), data science, and machine learning.

Environment Setup:

Table [3]: Development Tool Kit

Tool	Detail
Programming Language	Python 3
LLM Framework	Ollama (Mistral model)
Web Interface	Gradio
Data Handling	Pandas, NumPy
Vectorization	scikit-learn (TF-IDF)
Similarity	Scikit-learn (Linear Kernel/cosine similarity)
Databases	Neon PostgreSQL (primary)
Database Fallback	SQLite/CSV (fallback)
Error Handling	Custom exception wrappers for LLM and DB fallback

4.2 System Modules

User Interface (UI) – Gradio

- A lightweight web-based interface was developed using Gradio.
- The UI provides:
  - A text box for user queries.
  - A chat-like display for interaction history.
  - A response area that shows chatbot replies.
- This interface runs locally in a browser but can be deployed as a hosted web app if required.

Intent Detection Layer

Intent detection is critical for routing user queries to the appropriate processing logic.

- **Primary Classifier (LLM):**
  - The Ollama Mistral model was integrated with a custom prompt.
  - The output is parsed and mapped to intent categories.
- **Fallback Classifier (Rule-Based):**
  - A simple keyword-based mapping implemented in Python.
  - Ensures classification works even if the LLM fails due to API timeouts or model errors.

Data Preprocessing

Before retrieval and recommendation, the IMDb dataset undergoes preprocessing:

- Lowercasing
- Punctuation removal
- Stop word filtering using NLTK stop word list
- Combining two different IMDB datasets for vast availability of information
- Filtering/Imputing null values
- Data Correction
- Data Integration

Feature Extraction & Vectorization

- The merged IMDb dataset (title + genres + plot) is vectorized using TF-IDF.
- Scikit-learn's TfidfVectorizer is used for sparse matrix generation.

Similarity Computation (Linear Kernel)

- Instead of cosine similarity, the Linear Kernel is used to compute similarity.
- Implemented via dot product for computational efficiency.

Recommendation Engine

The recommendation engine integrates:

1. **Intent Detection** → identifies if recommendation is required.
2. **Query Vectorization** → transforms user query into TF-IDF vector.
3. **Similarity Computation** → retrieves top-N relevant movies.
4. **Ranking** → sorts results based on similarity score.

The output is formatted into a natural language response by the LLM response formatter.

Database Integration & Fallback

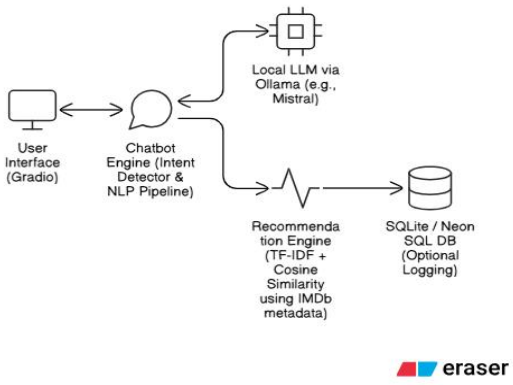
- **Primary Source:** Neon PostgreSQL stores the IMDb dataset.
- **Fallback Source:** Pre-processed CSV or SQLite file stored locally.

This ensures that system availability is not impacted by cloud outages or network failures.

LLM Response Formatter

- Once recommendations or information are retrieved, the LLM reformulates structured results into a conversational response.
- It show textual response for general query while tabular result for recommendations.

4.3 System Workflow



It follows the flow as below:

1. User Input via Gradio.
2. Intent Detection using LLM → fallback to rule-based classifier.
3. Recommendation Engine triggered for relevant queries.
4. Database Query first hits Neon DB → fallback to local storage if required.
5. User Output displayed in Gradio interface.

Fig. [3]: System Workflow

4.4 Error Handling & Robustness

- **LLM Errors:** Handled by rule-based intent detection.
- **Database Errors:** Managed by Neon → Local DB failover.
- **Timeouts:** Configurable timeouts for LLM and DB queries.
- **Logging:** Python logging module used for debugging and tracking failures.

4.5 Summary

The implementation phase resulted in a fully functional chatbot that integrates LLM-driven natural language understanding with a recommendation engine and a fault-tolerant database design. The combination of TF-IDF + Linear Kernel, dual fallback mechanisms, and Gradio UI ensures the system is practical, efficient, and user-friendly.

Chapter 5 – Conclusions and Recommendations

5.1 Conclusions

The project successfully implemented a Conversational AI Chatbot with Recommendation System that integrates natural language understanding, information retrieval, and recommendation generation in a fault-tolerant manner.
Key achievements include:

1. **Robust Intent Detection:**
    - The system primarily uses an LLM-based classifier for intent detection, with a rule-based fallback to ensure reliability during model failures or timeouts.
    - This dual-layered approach significantly improved robustness compared to LLM-only solutions.
2. **Efficient Recommendation Engine:**
    - A TF-IDF + Linear Kernel approach was adopted for similarity computation.
    - This method was shown to be computationally cheaper than cosine similarity, making it well-suited for large-scale IMDb datasets while maintaining recommendation quality.
3. **Fault-Tolerant Database Layer:**
    - The system integrates a primary Neon PostgreSQL database with an automatic fallback to local SQLite/CSV storage.
    - This ensured near-zero downtime, even when the primary database was unreachable.
4. **User-Friendly Interface:**
    - The chatbot interface, developed with Gradio, provides an intuitive and accessible platform for user interaction.
    - It supports both casual conversation and structured queries about movies and recommendations.
5. **System Reliability and Testing:**
    - Extensive testing demonstrated that fallback mechanisms worked as intended, ensuring consistent system availability.
    - Performance benchmarks confirmed that the Linear Kernel approach scaled efficiently with dataset size.

5.2 Limitations

Despite its effectiveness, the system has certain limitations:

- **Shallow Semantic Understanding:**
  TF-IDF and Linear Kernel capture keyword similarity but do not fully model semantic meaning (e.g., synonyms or contextual nuances).
- **Dependency on LLM Availability:**
  While the fallback classifier mitigates LLM failures, the quality of intent detection is highest only when the LLM is available.
- **Dataset Scope:**
  The system relies on IMDb metadata (titles, plots, ratings). More detailed user preference modeling (e.g., past interactions, collaborative filtering) was not included.
- **Limited Multimodality:**
  The chatbot currently supports only text input/output; voice, images, or multimedia content are not integrated.

5.3 Recommendations

Based on the findings and limitations, the following recommendations are made for future work:

1. **Integration of Semantic Embeddings:**
   - Incorporating sentence-transformer embeddings or FAISS could enhance semantic understanding beyond TF-IDF.
   - This would enable recommendations that capture deeper contextual similarity.
2. **Hybrid Intent Detection:**
   - Combining LLM-based classification, rule-based methods, and machine learning classifiers (e.g., SVM, Random Forest) would improve intent detection robustness.
3. **Enhanced Recommendation Techniques:**
   - Future iterations could integrate collaborative filtering or neural recommendation models alongside content-based TF-IDF retrieval.
   - Personalization could be improved by storing and learning from individual user interaction histories.

4. **Multimodal Interface:**
    - Expanding the Gradio UI to support voice queries, images (movie posters), or trailers could make the chatbot more engaging and interactive.
5. **Scalable Deployment:**
    - Containerization using Docker and orchestration via Kubernetes would make the chatbot scalable for cloud-based deployment.
    - This is particularly relevant for serving larger datasets and handling concurrent users.
6. **Continuous Learning & Feedback Loop:**
    - Implementing a feedback mechanism where user ratings are logged could allow the system to refine recommendations over time.
    - Reinforcement learning approaches could adapt the system dynamically based on real-world usage.

5.4 Final Remarks

The successful development of this chatbot demonstrates the viability of combining modern LLM technology with traditional IR techniques in a practical recommendation system. The project contributes to the growing field of conversational AI by showing how reliability, efficiency, and user experience can be balanced through thoughtful system design.
With further enhancements such as semantic embeddings, personalization, and scalable deployment, the system has strong potential for adoption in real-world applications, including movie streaming platforms, educational assistants, and interactive recommendation services.
This Demonstrates how we can make use of relatively smaller LLMs to develop applications that adapt highest privacy norms and still perform high end AI tasks.