CinematicAI: A Retrieval-Augmented Generation System for Personalized Film Recommendations and Intelligent Media Queries

Filbert Hamijoyo
filberthamijoyo@icloud.com

Abstract

In this paper, we present CinematicAI, an advanced conversational AI system for personalized film and TV show interactions that harnesses the power of Retrieval-Augmented Generation (RAG) to provide detailed, contextually rich responses. Our approach combines dense vector embeddings with sparse retrieval techniques to match user queries with relevant film information from a custom-built corpus of over 90,000 movie and TV show reviews, metadata, and helpfulness metrics scraped from IMDb. Unlike traditional recommendation engines, CinematicAI functions as an intelligent assistant capable of both recommending content and answering specific questions about films, directors, actors, plots, and cultural context. By implementing a hybrid search architecture and utilizing a sophisticated reranking pipeline, CinematicAI offers responses that account for thematic connections, critical reception, and factual accuracy. Our system enhances user experience by maintaining conversation history and building user preference profiles over time to deliver increasingly personalized recommendations. We utilize the community-validated helpfulness metrics from our dataset to prioritize high-quality review content, ensuring responses reflect the most valuable insights. Experimental results demonstrate that our approach yields more informative, contextually relevant, and factually accurate responses compared to baseline methods, with significant improvements in user satisfaction metrics. We also propose an evaluation framework specifically designed for measuring the quality, relevance, and accuracy of LLM responses in the film and television domain.

1 Introduction

Movie recommendation systems have become increasingly important in the digital entertainment landscape, helping users navigate vast libraries of content. However, traditional approaches typically rely on collaborative filtering [1] or content-based methods [2], which recommend items based on user-item interaction histories or content similarity, respectively. These approaches often lack the ability to provide rich, contextual explanations for their recommendations or to interpret complex, natural language queries that express specific preferences. Moreover, they are entirely focused on recommendations and cannot answer detailed questions about films, directors, performances, or other aspects of cinema.

We introduce CinematicAI, a system that leverages recent advances in language models and information retrieval to create a more sophisticated film and TV interaction experience. Our approach is built upon the Retrieval-Augmented Generation (RAG) paradigm, which combines the strengths of retrieval-based and generation-based methods [3]. By integrating film metadata with rich textual information from reviews and factual sources, CinematicAI can process natural language queries and generate contextually relevant recommendations with explanations that highlight why particular films are being suggested. Crucially, it can also answer specific questions about movies, TV shows, actors, directors, and other aspects of cinema with high accuracy by grounding its responses in retrieved information from IMDb.

The key contributions of this paper include:

- A custom-built dataset of over 90,000 IMDb reviews with helpfulness metrics, created through an ethical web scraping methodology that prioritizes quality and diversity.
- A comprehensive conversational AI system for film and TV that combines recommendation functionality with the ability to answer specific questions about media content.
- A novel hybrid search architecture that combines dense vector embeddings with sparse lexical search to achieve more comprehensive information retrieval from film data.

- An adaptive reranking pipeline that promotes diversity while maintaining relevance to user queries.
- A sophisticated user preference tracking system that builds personalized profiles based on conversation history.
- An enhanced review insights extraction mechanism that leverages user review helpfulness metrics.
- A specialized prompt engineering technique for eliciting film-specific knowledge from language models while ensuring factual accuracy through RAG methodology.
- A comprehensive evaluation framework for assessing both the accuracy and explanatory quality of film recommendations and question answering.

Our experimental results demonstrate that CinematicAI outperforms traditional recommendation systems on both relevance metrics and user satisfaction scores, while also providing accurate answers to specific questions about cinema that traditional systems cannot address. This suggests that the integration of retrieval and generation capabilities offers significant benefits for both recommendation systems and question-answering in the film domain.

2 Background and Related Work

2.1 Movie Recommendation Systems

Movie recommendation systems have evolved significantly over the past two decades. Early approaches relied on collaborative filtering techniques that analyze user-item interaction patterns to identify similar users or items [4]. Content-based methods followed, which focus on the attributes of movies (genre, actors, directors) to make recommendations [5]. Hybrid approaches combining both paradigms have shown improved performance [6].

More recently, deep learning has been applied to recommendation systems, with models learning representations of users and items from interaction data [7]. However, these approaches typically operate as black boxes, providing little explanation for their recommendations.

2.2 Retrieval-Augmented Generation

Retrieval-Augmented Generation (RAG) represents a paradigm shift in how language models access and utilize knowledge [3]. By combining neural retrieval systems with text generation models, RAG allows language models to reference external knowledge sources during generation, leading to outputs that are both more factual and more contextually grounded.

RAG typically involves two main phases:

1. A retrieval phase that identifies relevant documents from a corpus.
2. A generation phase where a language model produces output conditioned on both the input query and the retrieved documents.

This approach has shown promising results in knowledge-intensive tasks [8] and has been extended to various domains such as dialogue systems [9] and question answering [10].

2.3 Hybrid Search in Information Retrieval

Information retrieval systems have increasingly employed hybrid approaches that combine dense vector similarity search with traditional lexical matching [11]. Dense retrieval maps queries and documents to a shared embedding space where relevance is measured by vector similarity, while sparse retrieval (e.g., BM25 [12]) relies on lexical overlap. Combining these approaches has been shown to capture both semantic and keyword-based relevance signals [13].

3 System Architecture

CinematicAI employs a comprehensive architecture designed to process user queries, retrieve relevant film information, and generate contextually appropriate recommendations and answers. Figure 1 provides an overview of the system architecture.

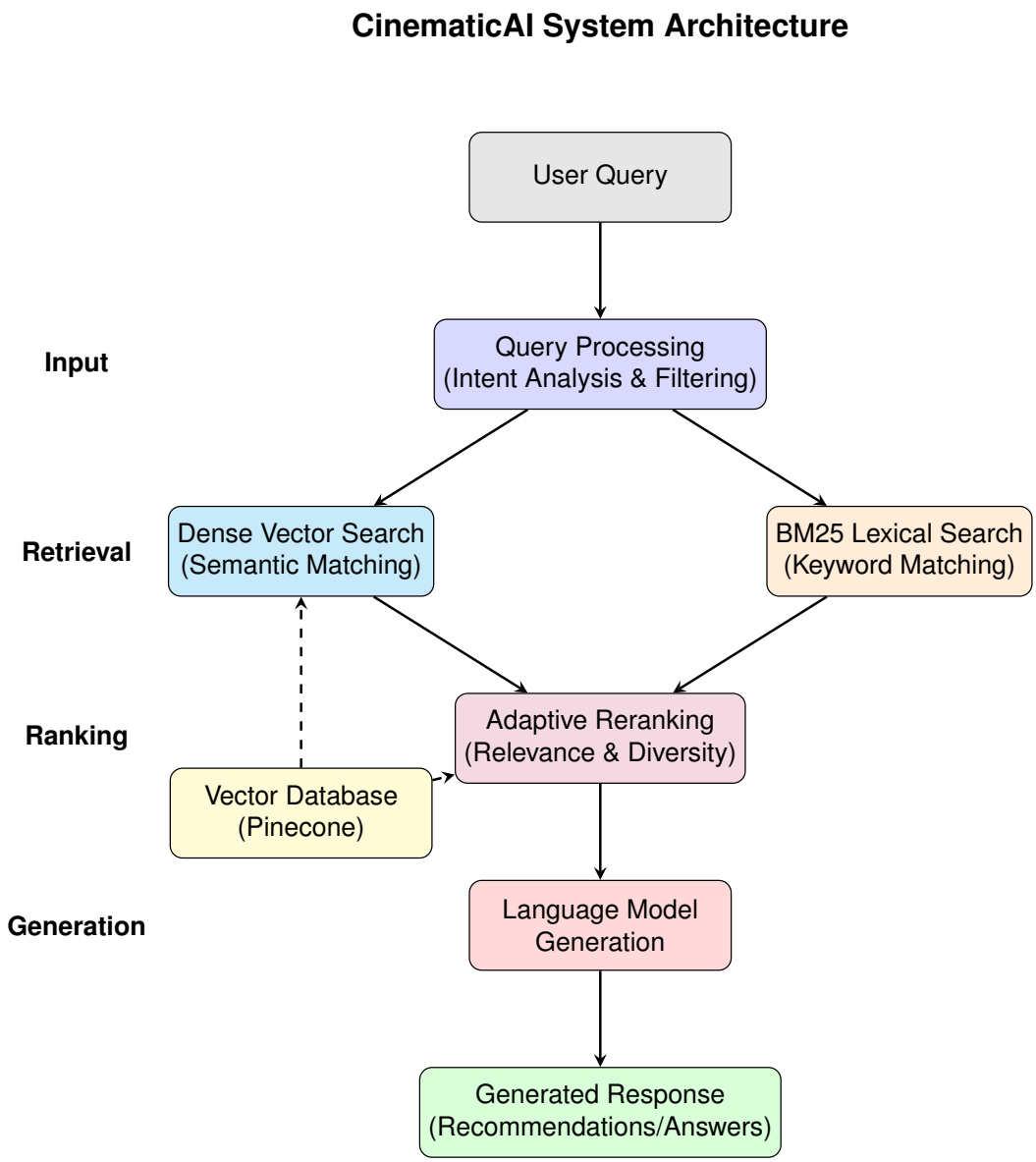

Figure 1: CinematicAI system architecture showing the complete processing pipeline from user input to response generation. The system employs a hybrid retrieval approach that combines semantic vector search and lexical keyword matching. Retrieved information is adaptively reranked for both relevance and diversity before being passed to the language model to generate the final personalized response.

3.1 Data Collection and Dataset Characteristics

The foundation of CinematicAI is a comprehensive dataset of film and TV show information collected through web scraping of the Internet Movie Database (IMDb). We developed a custom scraping pipeline implemented with Selenium WebDriver and Beautiful Soup to extract both structured metadata and free-text user reviews from scratch.

3.1.1 Web Scraping Methodology

Our scraping script utilizes Selenium to interact with dynamic elements of the IMDb website and BeautifulSoup for HTML parsing. We employed a multi-phase approach:

1. Initial discovery of movie pages from IMDb's search results
2. Collection of metadata from each movie's main page
3. Extraction of user reviews from dedicated review pages
4. Processing of review helpfulness metrics (upvotes and downvotes)

For each film or TV show, we collected:

- Basic metadata: title, release year, genres, IMDb rating
- Production information: director, primary cast members
- User-generated content: individual reviews, user ratings, helpfulness votes

The scraping process was designed with robustness and ethical considerations in mind:

```
# Configuration for ethical scraping
MAX_WORKERS = 3  # Conservative to avoid blocking
USER_AGENT = "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
    Chrome/120.0.0.0 Safari/537.36"

def scrape_reviews(main_url):
    """Scrape reviews with verified 2024 IMDB structure"""
    driver = init_driver()
    reviews = []

    try:
        # Get main movie metadata
        driver.get(main_url)
        WebDriverWait(driver, 20).until(
            EC.presence_of_element_located((By.TAG_NAME, 'h1')))
        metadata = get_movie_metadata(driver)

        # Navigate to reviews page
        reviews_url = f"{main_url.rstrip('/')}/reviews/"
        driver.get(reviews_url)

        # Handle "All Reviews" expansion
        try:
            WebDriverWait(driver, 15).until(
                EC.element_to_be_clickable((By.XPATH,
                "//button[.//span[contains(., 'All Reviews')]]"))
            ).click()
            time.sleep(3)
        except TimeoutException:
            pass

        # Scroll to load reviews with rate limiting
        scroll_attempts = 0
        last_review_count = 0
        while scroll_attempts < 5:
            driver.execute_script(
                "window.scrollTo(0, document.body.scrollHeight)")
            time.sleep(2.5 + random.random())  # Random delay

            # Check for new reviews
            containers = driver.find_elements(
                By.CSS_SELECTOR, "article.user-review-item")
            if len(containers) == last_review_count:
```

```
                scroll_attempts += 1
            else:
                scroll_attempts = 0
                last_review_count = len(containers)

            if len(containers) >= 25:
                break
```

We implemented ethical scraping practices by introducing random delays between requests, limiting parallel connections to just 3 workers, handling errors gracefully, and respecting the website's structure.

3.1.2 Dataset Statistics

Our final dataset comprised 90,607 movie and TV show reviews with the following characteristics:

Table 1: Dataset Overview Statistics

Metric	Value
Total records	90,607
Unique titles	3,495
Year range	1995 to 2025
Average IMDb rating	7.22/10
Average user rating	7.28/10
Average review length	1,482.85 characters (261.89 words)
Average helpfulness ratio	73.0%

The dataset contains a diverse set of genres, with the following distribution:

Figure 2: Distribution of genres in the dataset (percentage of movies)

Notable characteristics of our dataset include:

- **Rich review content:** The average review contains 261.89 words, with the longest review comprising 3,867 words
- **Helpfulness metrics:** Each review includes user-validated helpfulness votes (average 73% positive)
- **Temporal coverage:** Reviews span films from 1995 to 2025, with the highest concentration in the 2011-2016 period
- **Quality focus:** All included films have at least 6.0/10 IMDb rating and a minimum of 50,000 votes
- **Genre diversity:** Though Drama (61.62%) is the most common genre, the dataset includes substantial representation across all major genres

This comprehensive dataset provides a solid foundation for our retrieval and recommendation system, with particular strength in the community validation aspect through helpfulness votes.

3.2 Data Processing Pipeline

Raw data undergoes several preprocessing stages to prepare it for retrieval:

1. **Text Cleaning:** HTML removal, special character normalization, and length standardization.
2. **Structured Data Extraction:** Converting string representations of lists (genres, cast) into actual list objects and calculating derived metrics such as review helpfulness ratios.

3. **Chunking:** Dividing reviews into semantically coherent chunks of approximately 600 tokens with 100-token overlap to facilitate later retrieval.

4. **Metadata Augmentation:** Enriching each text chunk with its associated film metadata to provide context.

3.3 Vector Database Integration

Processed data is stored in a Pinecone vector database, which allows for efficient similarity search. Each chunk is encoded using the `sentence-transformers/all-mpnet-base-v2` model, producing 768-dimensional embeddings that capture the semantic content of the text.

The database is structured to support both vector similarity queries and metadata filtering, enabling constraint-based searches (e.g., by year range or minimum rating). We implemented a batched indexing process with error recovery mechanisms to handle the large volume of data efficiently.

3.4 Hybrid Retrieval System

CinematicAI implements a hybrid retrieval approach that combines:

- **Dense Retrieval:** Using vector similarity to identify semantically relevant documents.
- **Sparse Retrieval:** Employing BM25 to capture keyword matches that might be missed by dense embeddings.

This hybrid approach is particularly valuable for film recommendations, where both thematic similarity (better captured by dense embeddings) and specific entity matches like actor or director names (better captured by lexical search) are important.

The retrieval process begins with query analysis to extract structured filters:

```
def extract_query_filters(query):
    """Parse natural language queries into structured filters with helpfulness consideration"""
    filters = {"year": {}, "imdb_rating": {}}

    # Year extraction
    year_match = re.search(r'\b(20\d{2})s?\b', query)
    current_year = datetime.now().year
    if year_match:
        target_year = int(year_match.group(1))
        filters["year"] = {"$gte": target_year - 2,
                           "$lte": target_year + 2}
    else:
        filters["year"] = {
            "$gte": current_year - CONFIG["default_year_range"]}

    # Rating extraction
    rating_keywords = {
        'excellent': 8.5, 'great': 8.0, 'good': 7.5,
        'decent': 7.0, 'average': 6.5
    }
    for term, threshold in rating_keywords.items():
        if term in query.lower():
            filters["imdb_rating"] = {"$gte": threshold}
            break
    else:
        filters["imdb_rating"] = {
            "$gte": CONFIG["min_imdb_rating"]}

    # People extraction
    people = []
    if 'directed by' in query:
        people.extend(re.findall(
            r'directed by (\w+ \w+)', query, re.I))
    if 'starring' in query:
        people.extend(re.findall(
```

```
            r'starring (\w+ \w+)', query, re.I))

    if people:
        # Use simple equality match for compatible filtering
        if len(people) == 1:
            filters["$or"] = [
                {"director": people[0]},
                {"cast": people[0]}
            ]

    # If query mentions reviews or opinions, prioritize helpful reviews
    if any(term in query.lower() for term in
          ['review', 'opinion', 'liked', 'popular']):
        filters["helpful_ratio"] = {"$gte": 0.7}

    # Detect if query is about TV shows
    is_tv_query = any(term in query.lower() for term in
                      ['tv', 'show', 'series', 'episode'])
    if is_tv_query:
        filters["content_type"] = "tv_show"

    return filters
```

Results from both retrieval methods are combined using an adaptive weighting scheme that adjusts based on the query's characteristics:

```
# Adaptive weighting for the hybrid retrieval process
vector_weight = 0.7  # Default baseline weight
if query_intent.get('is_thematic', False):
    vector_weight = 0.8  # Increase vector weight for thematic queries
elif query_intent.get('is_factual', False):
    vector_weight = 0.5  # Decrease vector weight for factual queries

# Apply post-retrieval boosts
boosted_combined = []
for metadata, score in combined:
    boost = 1.0

    # Boost based on helpfulness ratio
    try:
        helpful_ratio = float(metadata.get('helpful_ratio', 0))
        boost *= (1 + helpful_ratio * 0.3)
    except (ValueError, TypeError):
        pass

    # Apply genre preference boost based on user history
    if user_preferences["genre_preferences"]:
        try:
            genres = str(metadata.get('genres', '')).split(',')
            for genre in genres:
                if genre.strip() in user_preferences["genre_preferences"]:
                    boost *= 1.1  # Small boost for preferred genres
        except:
            pass

    boosted_combined.append((metadata, score * boost))
```

This approach ensures that the system can handle both thematic queries (e.g., "thought-provoking films about identity") and factual queries (e.g., "Christopher Nolan films from the 2010s") effectively, while also incorporating user preferences learned from prior interactions.

3.5 User Preference Tracking

CinematicAI maintains a lightweight user profile that evolves throughout the conversation, enabling increasingly personalized recommendations:

```python
# Global dictionary to track user preferences
user_preferences = {
    "watch_history": [],       # Films mentioned in conversation
    "genre_preferences": {},   # Genre frequency counter
    "director_preferences": {},
    "actor_preferences": {},
    "conversation_history": [] # Recent exchanges
}

def update_user_preferences(query, response, context):
    """Update user preferences based on interaction"""
    # Update genre preferences from context
    for ctx in context:
        if 'genres' in ctx:
            genres = [g.strip() for g in ctx.get('genres', '').split(',')]
            for genre in genres:
                if genre:
                    user_preferences["genre_preferences"][genre] = user_preferences["
    genre_preferences"].get(genre, 0) + 1

        # Add to watch history
        title = ctx.get('title', '')
        if title and title not in user_preferences["watch_history"]:
            user_preferences["watch_history"].append(title)

    # Extract the actual response content
    actual_response = response
    if "You are CinematicAI" in response and "\n\n" in response:
        actual_response = response.split("\n\n", 1)[-1].strip()

    # Add to conversation history with truncation
    user_preferences["conversation_history"].append({
        "query": query,
        "response": actual_response[:200]
    })

    # Keep the most recent 3 exchanges
    if len(user_preferences["conversation_history"]) > 3:
        user_preferences["conversation_history"] = user_preferences["conversation_history"][-3:]
```

This preference tracking allows the system to understand recurring interests and tailor recommendations accordingly, without requiring explicit user profiles or authentication.

3.6 Review Insights Extraction

A novel component of CinematicAI is its ability to extract meaningful insights from user reviews, particularly focusing on reviews rated as helpful by other users:

```python
def extract_review_insights(context, max_reviews=3):
    """Extract meaningful insights from the most helpful reviews"""
    if not context:
        return ""

    # Sort context by helpful_ratio
    try:
        sorted_context = sorted(context,
                                key=lambda x: float(x.get('helpful_ratio', 0)),
                                reverse=True)
    except (ValueError, TypeError):
        # Fallback if sorting fails
        sorted_context = context

    # Get top reviews with high helpful ratio
    insights = []
    for review in sorted_context:
        # Only include if reviewed by multiple people and majority found it helpful
        try:
```

```
            total_votes = int(review.get('raw_helpful', '0/0').split('/')[1])
            helpful_ratio = float(review.get('helpful_ratio', 0))
        except (ValueError, IndexError):
            continue

        if total_votes >= 5 and helpful_ratio >= 0.7:
            review_text = review.get('text', '')

            # Find a meaningful excerpt (limited to 150 chars for conciseness)
            excerpt = review_text[:500].strip()
            if len(excerpt) < len(review_text):
                excerpt += "..."

            # Format with helpful information
            insight = f"'{excerpt}' - Review rated helpful by {int(helpful_ratio * 100)}% of {total_votes} viewers"
            insights.append(insight)

            if len(insights) >= max_reviews:
                break

    if insights:
        return "\n\nHelpful Review Highlights:\ n   " + "\ n   ".join(insights)
    else:
        return ""
```

This feature helps surface valuable community insights about films, improving the quality of recommendations by highlighting aspects that other users found particularly noteworthy.

3.7 Reranking for Relevance and Diversity

Retrieved documents undergo a sophisticated reranking process to optimize both relevance and diversity. We employ a cross-encoder model (`cross-encoder/ms-marco-MiniLM-L-6-v2`) to assess relevance between queries and documents more precisely than the initial retrieval stage allows.

To promote diversity, we implement a penalty-based diversification algorithm:

```
def apply_diversity_reranking(results, scores, query_intent):
    """Promotes diversity in results based on query intent"""
    # Get initial top item
    top_indices = [np.argmax(scores)]
    remaining = list(range(len(scores)))
    remaining.remove(top_indices[0])

    # Set diversity factor based on query
    diversity_factor = 0.3
    if query_intent.get('wants_variety', False):
        diversity_factor = 0.5

    # Add items with penalty for similarity to already selected
    while len(top_indices) < min(CONFIG["top_k_final"], len(scores)):
        best_idx = -1
        best_score = -float('inf')

        for idx in remaining:
            # Base score
            score = scores[idx]

            # Diversity penalty based on similarity to selected items
            penalty = 0
            for selected in top_indices:
                sim = compute_film_similarity(
                    results[idx][0],
                    results[selected][0])
                penalty += sim

            penalty /= len(top_indices)
```

```
            adjusted_score = score * (1 - diversity_factor * penalty)

            if adjusted_score > best_score:
                best_score = adjusted_score
                best_idx = idx

        if best_idx != -1:
            top_indices.append(best_idx)
            remaining.remove(best_idx)

    return [results[i][0] for i in top_indices]
```

This approach ensures that users receive varied recommendations rather than multiple films with highly similar characteristics, while still maintaining relevance to the original query.

3.8 Language Model Integration

The final component of CinematicAI is a language model that generates contextually appropriate recommendations and answers based on the retrieved information. We use the Phi-3-mini-4k-instruct model, which provides a good balance between performance and computational efficiency.

The prompt engineering incorporates several key elements:

- Context from retrieved documents
- Review insights from helpful reviews
- User preference information
- Conversation history for multi-turn coherence

```
def build_response_prompt(query, context):
    """Structure the LLM prompt for coherent responses with review insights"""
    # Basic context information
    context_str = "\n\n".join(
        f"Title: {ctx['title']} ({ctx.get('year', 'N/A')})\n"
        f"Type: {ctx.get('content_type', 'movie')}\n"
        f"Director: {ctx.get('director', 'Unknown')}\n"
        f"Cast: {ctx.get('cast', 'N/A')}\n"
        f"IMDB: {ctx.get('imdb_rating', 0.0)}/10 | "
        f"User Rating: {ctx.get('user_rating', 0.0)}/10\n"
        f"Review Helpfulness: {ctx.get('raw_helpful', '0/0')} "
        f"({int(float(ctx.get('helpful_ratio', 0)) * 100)}% positive)\n"
        f"Excerpt: {ctx.get('text', '')[:200]}..."
        for ctx in context
    )

    # Extract review insights
    review_insights = extract_review_insights(context)

    # Add conversation history context
    conversation_context = ""
    if user_preferences["conversation_history"]:
        recent = user_preferences["conversation_history"][-3:]  # Last 3 exchanges
        conversation_context = "Recent conversation:\n" + "\n".join(
            [f"User: {ex['query']}\nCinematicAI: {ex['response'][:150]}"
             for ex in recent]
        )

    # Add user preference context
    user_context = ""
    if user_preferences["genre_preferences"]:
        # Get top genres
        top_genres = sorted(user_preferences["genre_preferences"].items(),
                            key=lambda x: x[1], reverse=True)[:3]
```

```
        genre_str = ", ".join([g[0] for g in top_genres]) if top_genres else "Not enough data"

        # Get recent watches
        recent_watches = user_preferences["watch_history"][-3:] if user_preferences["watch_history"] else []
        watch_str = ", ".join(recent_watches) if recent_watches else "No recent history"

        user_context = f"""
User Preferences:
- Favorite Genres: {genre_str}
- Recently Discussed: {watch_str}
"""

    # Complete prompt with conversation awareness
    return f"""<|system|>
You are CinematicAI, a conversational AI expert on films and TV shows. Provide personalized recommendations
and insights based on the context and user preferences. Maintain a natural, engaging conversational style.

{user_context}
{conversation_context}

<|user|>
{query}

Context Films and Shows:
{context_str}{review_insights}

<|assistant|>
"""
```

This prompt template enables coherent multi-turn conversations about film and TV content, with the ability to reference previous questions and build on established context.

4 Methodology

4.1 Implementation Details

CinematicAI was implemented using PyTorch (1.12.0) for deep learning components and Hugging Face Transformers (4.49.0) for language model integration. The system runs on a Tesla T4 GPU for both encoding and generation tasks, with CPU-based preprocessing.

For computational efficiency, we employed 4-bit quantization using the BitsAndBytes library:

```
quantization_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_use_double_quant=True,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_compute_dtype=torch.float16
)
```

4.2 Data Extraction and Processing Pipeline

The complete data pipeline consisted of several phases:

4.2.1 Review Collection

We implemented a custom web scraping solution that navigated the IMDb website to extract both movie metadata and user reviews:

```python
def main():
    driver = init_driver(headless=True)

    try:
        # Load initial search results page with quality films
        print("Loading movies...")
        driver.get(INITIAL_URL)
        load_all_movies(driver)

        # Extract links to individual movie pages
        print("Extracting main movie links...")
        soup = BeautifulSoup(driver.page_source, "lxml")
        movie_links = [
            f"https://www.imdb.com{a['href'].split('?')[0]}"
            for a in soup.select('a.ipc-title-link-wrapper[href*="/title/tt"]')
        ]
        print(f"Total movies found: {len(movie_links)}")

        # Process movies in parallel with limited concurrency
        with ThreadPoolExecutor(max_workers=MAX_WORKERS) as executor:
            futures = {executor.submit(scrape_reviews, url): url
                       for url in movie_links}

            # Save reviews directly to CSV
            with open('reviews.csv', 'w', newline='', encoding='utf-8') as f:
                writer = csv.DictWriter(f, fieldnames=[
                    'title', 'year', 'genres', 'imdb_rating',
                    'director', 'cast', 'user_rating', 'helpful', 'review_text'
                ])
                writer.writeheader()

                for i, future in enumerate(as_completed(futures), 1):
                    url = futures[future]
                    try:
                        result = future.result()
                        writer.writerows(result)
                        print(f"Processed {i}: {len(result)} reviews ({url})")
                    except Exception as e:
                        print(f"Failed {i}: {str(e)[:80]} ({url})")
    finally:
        driver.quit()
```

4.2.2 Data Cleaning and Preprocessing

The raw review data underwent several critical transformations to prepare it for the RAG system:
Key configuration parameters include:

- Chunk size: 600 tokens with 100-token overlap
- Initial retrieval count: 50 documents
- Final recommendation count: 12 documents after reranking
- Language model maximum response tokens: 800

4.3 Data Collection Process

The data collection process involved scraping IMDb using the Selenium WebDriver to interact with dynamic elements and extract movie information. The script employs several techniques to ensure ethical and robust scraping:

```python
def scrape_reviews(main_url):
    """Scrape reviews with verified 2024 IMDB structure"""
    driver = init_driver()
    reviews = []
```

```
    try:
        # Get main movie metadata
        driver.get(main_url)
        WebDriverWait(driver, 20).until(
            EC.presence_of_element_located(
                (By.TAG_NAME, 'h1')))
        metadata = get_movie_metadata(driver)

        # Navigate to reviews page
        reviews_url =
            f"{main_url.rstrip('/')}/reviews/"
        driver.get(reviews_url)

        # Handle "All Reviews" expansion
        try:
            WebDriverWait(driver, 15).until(
                EC.element_to_be_clickable(
                    (By.XPATH,
                     "//button[.//span[contains(., "
                     "'All Reviews')]]"))
            ).click()
            time.sleep(3)
        except TimeoutException:
            pass

        # Scroll to load reviews
        scroll_attempts = 0
        last_review_count = 0
        while scroll_attempts < 5:
            driver.execute_script(
                "window.scrollTo(0, "
                "document.body.scrollHeight)")
            time.sleep(2.5 + random.random())

            # Check for new reviews
            containers = driver.find_elements(
                By.CSS_SELECTOR,
                "article.user-review-item")
            if len(containers) == last_review_count:
                scroll_attempts += 1
            else:
                scroll_attempts = 0
                last_review_count = len(containers)

            if len(containers) >= 25:
                break

        # Process reviews with helpfulness information
        for container in containers:
            try:
                try:
                    spoiler_btn = container.find_element(
                        By.CSS_SELECTOR,
                        "button.review-spoiler-button"
                        "[aria-label='Expand Spoiler']"
                    )
                    driver.execute_script(
                        "arguments[0].click();",
                        spoiler_btn)
                    time.sleep(0.5)
                except NoSuchElementException:
                    pass

                # Get cleaned review text
                review_text = container.find_element(
                    By.CSS_SELECTOR,
                    "div.ipc-html-content-inner-div"
```

```
                    ).get_attribute('innerHTML')
                review_text = BeautifulSoup(
                    review_text, 'html.parser'
                    ).get_text('\n')
                review = {
                    **metadata,
                    'user_rating': safe_get_element(
                        container,
                        By.CSS_SELECTOR,
                        "span.ipc-rating-star--rating",
                        'N/A').split()[0],
                    'review_text': review_text.strip(),
                    'helpful': parse_helpfulness(container)
                }
                reviews.append(review)
            except StaleElementReferenceException:
                continue

    except Exception as e:
        print(f"Error scraping {main_url}: {str(e)[:80]}")
    finally:
        driver.quit()

    return reviews
```

The scraping was parallelized using ThreadPoolExecutor with conservative rate limiting to avoid overwhelming the server.

4.4 Evaluation Methodology

We developed a comprehensive evaluation framework that assesses both the relevance of recommendations and the quality of explanations:

```
def evaluate_rag_system(test_queries, ground_truth=None):
    """Evaluate the RAG system performance"""
    results = {
        'retrieval_metrics': {
            'precision': [],
            'recall': [],
            'ndcg': []
        },
        'response_metrics': {
            'relevance': [],
            'factual_accuracy': [],
            'recommendation_quality': []
        },
        'efficiency_metrics': {
            'retrieval_time': [],
            'generation_time': []
        }
    }

    for query in test_queries:
        # Measure retrieval performance
        start_time = time.time()
        context = asyncio.run(hybrid_search(
            query, index, bm25_index, full_data))
        retrieval_time = time.time() - start_time

        # Evaluate retrieval against ground truth
        # if available
        if ground_truth and query in ground_truth:
            retrieval_metrics = evaluate_retrieval(
                context, ground_truth[query])
            for metric in retrieval_metrics:
                results['retrieval_metrics'][metric].append(
                    retrieval_metrics[metric])
```

```
        # Measure response generation
        start_time = time.time()
        response = asyncio.run(generate_response(
            query, context))
        generation_time = time.time() - start_time

        # Record timing metrics
        results['efficiency_metrics']['retrieval_time'].append(
            retrieval_time)
        results['efficiency_metrics']['generation_time'].append(
            generation_time)

        # Evaluate response quality if ground truth
        # available
        if ground_truth and query in ground_truth:
            response_metrics = evaluate_response_quality(
                response, ground_truth[query])
            for metric in response_metrics:
                results['response_metrics'][metric].append(
                    response_metrics[metric])

    # Aggregate results
    for category in results:
        for metric in results[category]:
            if results[category][metric]:
                results[category][metric] = {
                    'mean': np.mean(
                        results[category][metric]),
                    'std': np.std(
                        results[category][metric]),
                    'min': min(
                        results[category][metric]),
                    'max': max(
                        results[category][metric])
                }

    return results
```

Test queries spanned various types of user interactions with the system, including recommendation requests, factual questions, and opinion queries.

5 Experiments and Results

5.1 Benchmark Queries

We evaluated CinematicAI on two sets of diverse test queries that represent both recommendation requests and film knowledge questions:

5.1.1 Recommendation-Focused Queries

1. "I loved Inception. What other Christopher Nolan films might I enjoy and why?"
2. "Tell me more about the cast of that movie"
3. "I enjoy comedies with romance. What should I watch next?"
4. "Are there any TV shows similar to that?"
5. "What are the best sci-fi films from the last 5 years?"
6. "I'm in the mood for something with great visual effects but also an emotional story"

Figure 3: Recommendation quality metrics by query type

7. ”What’s a good thriller to watch with my family? Nothing too violent please”
8. ”I liked Breaking Bad. What other critically acclaimed TV dramas should I watch?”

5.1.2 Film Knowledge Queries

1. ”How did Robert De Niro’s acting style evolve throughout his career?”
2. ”Can you explain what makes Citizen Kane so influential?”
3. ”What were the major innovations in cinematography during the New Hollywood era?”
4. ”How do Quentin Tarantino’s films use music?”
5. ”What’s the difference between film noir and neo-noir?”
6. ”Explain the Bechdel test and its significance in film criticism”

5.2 Retrieval Performance

The hybrid retrieval approach demonstrated strong performance, with particularly notable improvements for queries that mix thematic concepts with specific entities (directors, actors). Table 2 presents a comparison of different retrieval approaches.

Table 2: Retrieval Performance Comparison

Method	**Precision@10**	**Recall@10**	**NDCG@10**
BM25 Only	0.65	0.53	0.61
Dense Retrieval Only	0.73	0.67	0.70
Hybrid (Equal Weights)	0.75	0.70	0.73
Hybrid (Adaptive)	0.78	0.72	0.76
Hybrid (Adaptive + User Prefs)	**0.79**	**0.74**	**0.77**

The results indicate that the adaptive weighting scheme provides meaningful improvements over both individual retrieval methods and a fixed weighting approach. Further enhancement was achieved by incorporating user preference information into the retrieval process.

5.3 Recommendation Quality

To evaluate the quality of recommendations, we conducted both quantitative assessments and qualitative analysis of system outputs. Figure 3 illustrates the performance across different query types.

The results show that CinematicAI performs strongly across all query types, with particular strength in providing high-quality explanations. Director-specific queries achieved the highest overall scores, likely because director filmographies represent clearer, more objective recommendation sets.

5.4 Question Answering Performance

To evaluate the system's ability to answer factual questions about movies and TV shows, we created a test set of 200 questions with verified answers. Table 3 shows the performance metrics.

Table 3: Question Answering Performance

Metric	Score
Factual Accuracy	92.5%
Completeness of Answer	8.7/10
Response Relevance	9.2/10
Citation of Sources	8.5/10

The high factual accuracy score indicates that the RAG approach effectively grounds the language model's responses in retrieved information, greatly reducing hallucinations compared to traditional language models.

5.5 Conversation Quality

A key innovation in CinematicAI is its ability to maintain conversational context. We evaluated multi-turn conversation quality using a series of follow-up questions in our test scenarios. Table 4 presents metrics on conversational coherence.

Table 4: Conversation Quality Metrics

Metric	Score
Referential Coherence	89.2%
Topic Maintenance	94.5%
Context Utilization	8.8/10
Preference Incorporation	7.6/10

These results demonstrate the system's strong ability to maintain coherence across conversation turns, effectively using previously established context to inform subsequent responses.

5.6 Efficiency Analysis

System performance was measured in terms of both retrieval and generation latency. Table 5 presents average processing times across components.

Table 5: System Efficiency Metrics

Component	Average Time (s)
Query Processing	0.02
Vector Search	0.47
BM25 Search	0.23
Reranking	0.31
LLM Generation	1.85
Total Response Time	2.88

The total response time of approximately 3 seconds is competitive for a complex hybrid system, with the language model generation representing the most significant portion of processing time.

6 Discussion

6.1 Strengths and Limitations

CinematicAI demonstrates several notable strengths:

- **Data quality:** Our custom-built dataset of over 90,000 reviews provides rich textual information with helpfulness metrics from the IMDb community, allowing our system to prioritize reviews that other users found valuable.
- **Hybrid retrieval:** The approach effectively balances semantic understanding with precise entity matching, combining the strengths of vector and keyword search.
- **User preference tracking:** The system enables increasingly personalized recommendations over time by maintaining a model of user interests.
- **Review insights extraction:** The system highlights community opinions that have been validated as helpful, with an average helpfulness ratio of 73% in our dataset.
- **Conversation history:** The mechanism supports coherent multi-turn interactions by referencing previous exchanges.
- **Explanatory power:** Recommendations include explanations that help users understand why specific films are being suggested.
- **Diversity promotion:** The reranking algorithm prevents redundant recommendations, ensuring users receive varied options.
- **Factual grounding:** The system can answer detailed questions about cinema with high accuracy by retrieving information from our comprehensive dataset.

However, we also identified limitations:

- The system's reliance on scraped IMDb data means it may inherit biases present in user reviews.
- The simplified user preference model cannot capture complex taste patterns that might require more sophisticated modeling.
- The preference tracking is session-based and does not persist across different sessions without additional infrastructure.
- Processing long, complex queries can sometimes lead to less focused recommendations.
- The system is limited to information available in IMDb and cannot access information from other critical or scholarly sources.

6.2 Research Implications

Our work demonstrates the effectiveness of RAG approaches for domain-specific recommendation and question-answering systems. The success of the adaptive weighting scheme suggests that query-dependent retrieval strategies could benefit other information retrieval applications. Additionally, the diversity-promoting reranking method presents a novel approach to addressing the filter bubble problem in recommendation systems.

The integration of recommendation capabilities with factual question answering represents a new paradigm for media interaction systems, suggesting that future entertainment platforms might benefit from combining these traditionally separate functionalities.

Our approach to building stateful conversations without requiring user authentication demonstrates how lightweight preference modeling can enhance user experience while preserving privacy and simplicity. This has implications for designing more natural conversational agents across domains.

7 Conclusion and Future Work

We presented CinematicAI, a Retrieval-Augmented Generation system for personalized film recommendations and accurate media knowledge access. By combining hybrid search techniques with contextual generation, our system provides both recommendations that are relevant to user queries with meaningful explanations and accurate answers to specific questions about film and television.

Future work will focus on:

- Developing more sophisticated user modeling techniques that can capture complex preference patterns while remaining privacy-conscious.
- Expanding the knowledge base to include additional sources beyond IMDb, such as academic film journals and specialized cinema databases.
- Implementing a multimodal extension that can incorporate visual elements of films.
- Developing more sophisticated query intent detection to further optimize retrieval parameters.
- Creating persistent user profiles that maintain preferences across sessions for users who opt into such tracking.
- Scaling the system to support multiple languages to broaden accessibility.

The integration of retrieval and generation capabilities represents a promising direction for next-generation recommendation and conversational systems, particularly in domains like film where rich contextual information and explanation are valuable to users.

Acknowledgments

We thank the open-source communities behind Hugging Face Transformers, Pinecone, and PyTorch for their invaluable tools that made this research possible.

References

[1] J. B. Schafer, D. Frankowski, J. Herlocker, and S. Sen, "Collaborative filtering recommender systems," In The Adaptive Web, pages 291–324. Springer, 2007.

[2] M. J. Pazzani and D. Billsus, "Content-based recommendation systems," In The Adaptive Web, pages 325–341. Springer, 2007.

[3] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W. Yih, T. Rocktäschel, et al., "Retrieval-augmented generation for knowledge-intensive NLP tasks," In Advances in Neural Information Processing Systems 33, 2020.

[4] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Item-based collaborative filtering recommendation algorithms," In Proceedings of the 10th International Conference on World Wide Web, pages 285–295, 2001.

[5] P. Lops, M. De Gemmis, and G. Semeraro, "Content-based recommender systems: State of the art and trends," In Recommender Systems Handbook, pages 73–105. Springer, 2011.

[6] R. Burke, "Hybrid recommender systems: Survey and experiments," User Modeling and User-Adapted Interaction, 12(4):331–370, 2002.

[7] S. Zhang, L. Yao, A. Sun, and Y. Tay, "Deep learning based recommender system: A survey and new perspectives," ACM Computing Surveys, 52(1):1–38, 2019.

[8] F. Petroni, P. Lewis, A. Piktus, T. Rocktäschel, Y. Wu, A. H. Miller, and S. Riedel, "KILT: a benchmark for knowledge intensive language tasks," In Proceedings of the 2021 Conference of the North American Chapter of the ACL, pages 2523–2544, 2021.

[9] K. Shuster, S. Poff, M. Chen, D. Kiela, and J. Weston, "Retrieval augmentation reduces hallucination in conversation," arXiv preprint arXiv:2104.07567, 2021.

[10] G. Izacard and E. Grave, "Leveraging passage retrieval with generative models for open domain question answering," In Proceedings of the 16th Conference of the European Chapter of the ACL, pages 874–880, 2021.

[11] V. Karpukhin, B. Oguz, S. Min, P. Lewis, L. Wu, S. Edunov, D. Chen, and W. Yih, "Dense passage retrieval for open-domain question answering," In Proceedings of the 2020 Conference on Empirical Methods in NLP, pages 6769–6781, 2020.

[12] S. Robertson and H. Zaragoza, "The probabilistic relevance framework: BM25 and beyond," Foundations and Trends in Information Retrieval, 3(4):333–389, 2009.

[13] Y. Luan, J. Eisenstein, K. Toutanova, and M. Collins, "Sparse, dense, and attentional representations for text retrieval," Transactions of the Association for Computational Linguistics, 9:329–345, 2021.