Pusan National University

Computer Science and Engineering

Technical Report 2024-10

대형 언어모델을 활용한 추천 시스템의 개선 방안

저자1 박지환

저자2 장재혁

저자3 하현진

지도교수 조준수

목 차

1. 서론 1

1.1. 연구 배경 1

1.2. 주요 문제점 분석 2

1.3. 연구 목표 3

2. 연구 배경 3

2.1. 추천 시스템 3

2.2. 대형 언어 모델 5

2.3. 프론트엔드 6

3. 연구 내용 8

3.1. 구성원 별 역할 8

3.2. 데이터 확보 및 전처리 9

4. 연구 결과 분석 및 평가 25

5. 결론 및 향후 연구 방향 30

6. 참고 문헌 30

1. 서론

1.1. 연구 배경

2022년 OpenAI사의 대화형 인공지능 모델인 'ChatGPT'가 공개된 이래로 대형 언어 모델(Large Language Model, 이하 LLM) 분야는 끊임없이 발전을 거듭하고 있다. 더 빠른 속도, 더 강력한 성능, 더 저렴한 비용을 주요 키워드로 내세우며 발전해 온 LLM들은 이제는 사람들의 일상 생활 속에서도 어렵지 않게 찾을 수 있게 되었다.

이러한 LLM의 강력한 추론 능력을 활용할 수 있는 여러 방안 중 추천 시스템과 LLM을 결합한 '대화형 추천 시스템(Conversational Recommender System, 이하 CRS)'의 개념이 제시되었다. 기존 추천 시스템의 한계를 LLM을 활용하여 극복하고, 사용자와의 대화를 통하여 더 세밀하고 개인화된 추천을 제공한다는 목표를 가진 CRS는 현재도 활발한 연구가 이루어지고 있는 분야이다.

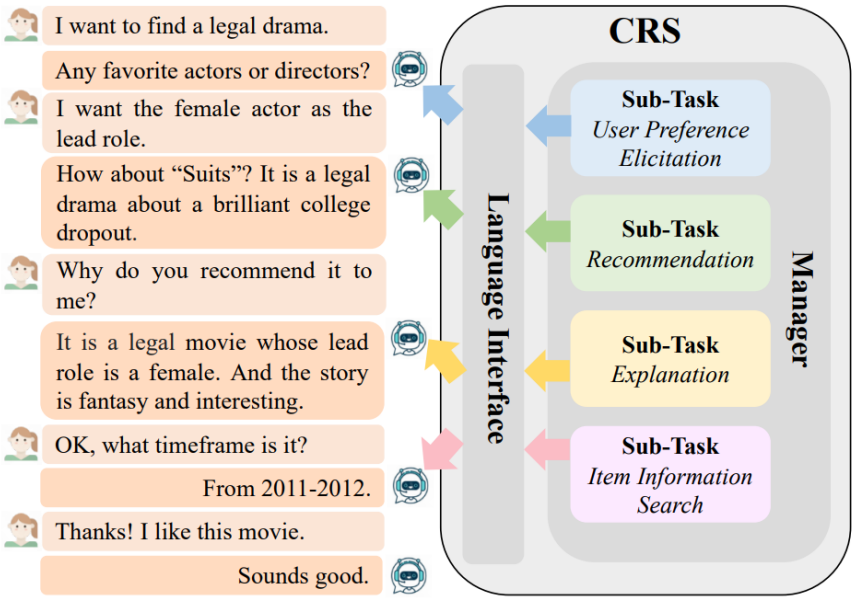

그림 1. 대화형 추천 시스템(CRS)의 예시 [1]

기존 추천 시스템의 경우 클릭이나 사용자의 시청/구매 기록 등의 암시적 상호작용 신호에 크게 의존한다. 이는 정보가 부족하거나 아예 없는 새로운 사용자나 아이템에 대한 정확한 추천을 어렵게 하는 cold-start 문제가 발생하는 원인이 된다. 또한, 암시적 신호만으로 사용자에게 추천을 진행하는 방식은 사용자의 활동 맥락을 정확하게 반영하기 어렵기 때문에, 추천 결과의 신뢰도가 높지 않을 가능성이 있다. 마지막으로, 대다수의 추천 시스템이 결과만을 산출하는 'point-and-click' 인터페이스를 기반으로 한 모델인 경우가 많기 때문에 추천 결과에 대한 충분한 근거를 제시하지 못하는 explainability 부족 및 사용자 피드백 반영의 부족 문제

또한 발생하게 된다. [2]

이와 같은 문제들을 해결하기 위해 LLM을 도입하여 사용자와의 실시간 대화 인터페이스를 통하여 명시적인 상호작용 및 선호 신호를 획득하고, 이를 기반으로 추천 근거를 생성하는 등 LLM의 추론 및 언어 생성 능력을 최대한으로 활용하는 것이 CRS의 주 목표이다.

이번 연구에서는 LLM과 추천 시스템을 결합하여 사용자의 발화 내용을 기반으로 개인화된 영화 추천을 제공하는 CRS를 개발한다. 데이터셋의 가공에서부터 사용자와의 상호작용 및 추천 결과 출력까지 전체 기능과 구조를 구현하고 웹 환경으로 서비스한다.

1.2. 주요 문제점 분석

1.2.1. LLM 적용 범위의 모호성 문제

CRS는 아직 연구가 진행되고 있는 분야인 만큼 추천 시스템과 LLM을 결합함에 있어서 정확히 어떠한 부분에서 LLM이 적용이 되어야 하는지, 프롬프트의 설계는 어떤 식으로 이루어져야 하는지에 대한 명확한 가이드라인이 부족하다. 따라서 여러 논문을 참조하여 기본적인 틀을 설계한 뒤, 여러 가지 시도를 통하여 전체적인 시스템의 정확도와 성능을 개선하고자 한다.

1.2.2. 자원 요구량 과다의 문제

추천 시스템에서 사용자의 정보를 입력받아 최종 추천 결과를 산출하기까지에 소요되는 시간 또한 중요한 평가 지표의 일종이다. 동시에 구동되는 추천 모듈과 LLM의 실시간 상호 입출력을 기반으로 하는 웹 서비스의 특성 상 일반적인 환경에서 연산 자원 및 시간이 많이 요구되는데, 모델 및 추천 시스템 전체에서의 경량화 작업을 통하여 이를 최소화하고자 한다.

1.2.3. 평가 방법론의 한계

정답이 레이블로 주어져 있는 기존 추천 시스템과 다르게, 대화형 추천 시스템의 경우에는 실시간으로 변동하는 사용자의 의향과 상황 맥락을 파악하여 사용자가 납득 가능한 추천 결과를 도출하여야 한다. 이에 따라 기존 추천 시스템을 평가하는 평가 지표와 방식이 CRS 의 정확한 성능을 평가하기에는 부적절할 수 있다. 이에 따라 새로운 평가 지표에 대한 연구가 수행되고 있고 실제로 제안된 평가 지표가 존재하나[3], 우선 테스트 단계에서 사용자의 발화 내용과 추천 근거의 explanation을 대조하여 논리적으로 연결되는 근거가 제시되었는지에 중점을 두고 테스트를 진행하고자 한다.

1.3. 연구 목표

1.3.1. 추천 시스템 - LLM 연동 구조 정립 및 개발

LLM을 추천 시스템에 결합하는 방식 및 범위에 대한 방법론적인 연구는 많이 선행되었다. 그러나 구체적인 구현에 대해서는 명확한 자료의 양이 부족하기 때문에, 기존 연구 자료를 기반으로 하여 추천 시스템과 LLM의 상호 입출력 연동 절차 및 전체적인 시스템의 구조를 정립하고 구현하는 것을 목표로 한다.

1.3.2. 추천 시스템 및 LLM 모듈에 대한 경량화 작업 수행

모델 선정과 추천 시스템 구축 과정에서 여러 가지 경량화 방식을 적용한 뒤, 성능과 자원 요구량의 균형을 갖춘 시스템을 구축하는 것을 목표로 한다.

1.3.3. 최적의 추천 제공 서비스

사용자의 발화 내용에서 유용한 정보를 추출하고, 이를 기반으로 추천을 진행함으로써 각 사용자의 선호와 맥락에 일치하는 최적의 추천 및 근거를 산출하는 서비스를 구축하는 것을 목표로 한다.

2. 연구 배경

2.1. 추천 시스템

2.1.1. 콘텐츠 기반 필터링

콘텐츠 기반 필터링(Content-based Filtering)은 추천 시스템 방식의 일종으로, 아이템의 특성과 사용자의 선호 정도를 분석 및 비교하여 추천을 제공하는 방식이다. 사용자가 선호하였던 아이템의 특성을 기반으로 새로운 아이템을 추천한다. 이러한 방식의 특성으로 인하여 다른 사용자의 데이터가 없는 상황에도 현재 사용자의 선호를 분석, 추천을 진행할 수 있고, 추천에 대한 근거를 생성하기에도 적합하다.

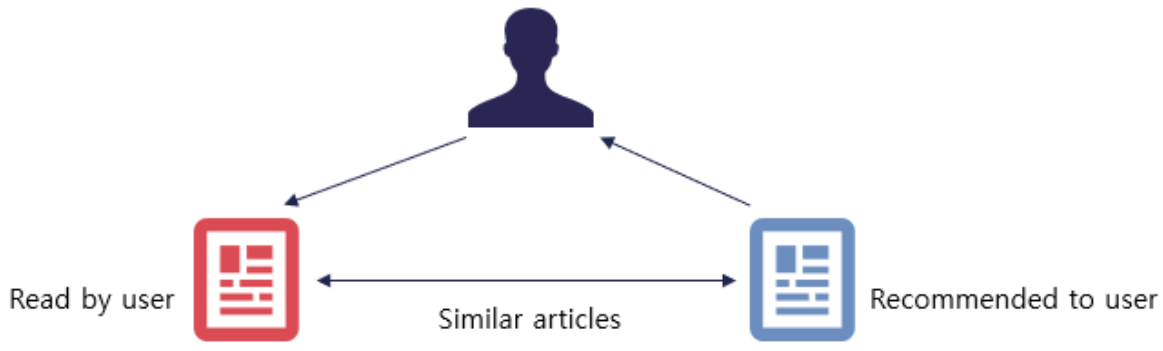

그림 2. 콘텐츠 기반 필터링의 예시 : 기사 추천

본 연구에서는 사용자 개인의 선호 정보에 중점을 두면서 LLM의 추론 기능의 지원을 받는 개인화 추천 시스템을 구축하였다. 시스템의 기능적 목표를 보았을 때에 아이템에 대한 타 사용자들의 평가 및 해당 사용자들의 특성과 비교(협업 필터링, Collaborative Filtering)하는 방식보다는 사용자의 선호 특성과 아이템의 특성의 유사도를 측정하여 추천하는 방식의 중요도가 상대적으로 더 높다고 판단하였다. 이에 따라 콘텐츠 기반 필터링 방식을 기조로 하면서 타 사용자들의 평점 정보에 약소한 가중치를 두어 추천에 영향을 주는 방식을 차용, 콘텐츠 기반 필터링을 중심으로 한 하이브리드 추천 시스템을 구현하였다.

2.1.2. 코사인 유사도

공간 내 두 벡터 사잇각의 코사인값을 활용하여 두 벡터 간의 유사성을 측정하는 방식이다. 유사도 산출 식은 아래와 같다.

$$\text{Cosine Similarity } = \frac{X \cdot Y}{||X||_2 \cdot ||Y||_2}, \text{ where } ||X||_2 \text{ is L2 norm}$$

식의 결과값은 [-1,1] 범위 내에 존재하게 되는데 두 벡터의 방향이 완전히 동일한 경우, 즉 일치한 경우에 코사인 유사도 값은 1을 갖게 되고, 두 벡터의 방향이 완전히 반대일 경우에는 -1의 값을 갖게 된다. 위 식을 이용, 벡터 간의 유사성을 특정 범위 내의 수치로 치환하는 과정을 통하여 추천 시스템 내 특성 벡터 활용에서의 용이성을 확보할 수 있다.

2.1.3. MLB

MLB는 Multi-Label Binarizer의 약자로 다중 레이블을 갖는 데이터를 이진 행렬로 변환하는 도구이다. 각 레이블에 대한 열을 생성한 뒤, 인스턴스가 특정 레이블에 해당된다면 1의 값을, 그렇지 않을 경우 0을 할당하는 작업을 수행한다. 이는 원-핫 인코딩(One-hot Encoding) 방식과 유사하게 볼 수 있는데, 원-핫 인코딩이 수행된 인스턴스는 1의 값을 갖는 열, 즉 해당되는 클래스가 한 가지인 것과 달리 MLB는 다중 레이블 문제 환경 내에서 사용되기 때문에 결과 행에서 1의 값을 갖는 열이 하나 이상이라는 차이점이 있다. 이러한 도구를 활용하여 영화와 같이 배우, 장르 등 동시에 여러 값을 갖는 아이템들을 추천 알고리즘에 입력되기 적합한 형태로 변환할 수 있다.

Multi-Label Problem:		Output vector:		
Instance	Classes	A	B	C
1	A, B	1	1	0
2	A	1	0	0
3	A, B	1	1	0
4	C	0	0	1

그림 3. Multi-Label Problem의 예시

2.2. 대형 언어 모델

2.2.1. 양자화(Quantization)

LLM(Large Language Model)에서의 양자화(Quantization)는 모델의 크기를 줄이고 효율성을 높이기 위한 기술이다. 양자화는 LLM의 가중치(weights)와 활성화(activations)를 고정밀도 값에서 저정밀도 값으로 변환하는 모델 압축 기술이다. 예를 들어, 32비트 부동소수점 숫자를 8비트 정수로 변환하는 것이 일반적인 양자화 과정이다. 메모리 효율성, 저장 공간 절약, 에너지 효율성 등 자원을 절약할 수 있으며 추론 속도에서 이점을 얻는다. 또한, 자원이 제한된 환경에서도 실행할 수 있게 된다.

본 연구에서는 높은 한국어 처리 능력 요구 및 한정된 연산 자원량 등의 문제들을 고려하여 기본 모델로 EEVE-Korean-Instruct-10.8B-v1.0-Q5_K_M.gguf을 채택하였다. 해당 모델은 upstage에서 개발한 SOLAR-10.7B-v1.0모델을 기반으로, 야놀자에서 한국어 파인튜닝을 추가 진행하여 제작한 모델이다. 연산 자원 요구량을 최소화하기 위하여, 해당 모델을 양자화한 후, 해당 버전을 최종 모델로 선정하였다. Q5_K_M은 5비트 양자화, K_quant, Medium을 의미하여 적절한 모델 크기와 성능을 보여주는 조합이다.

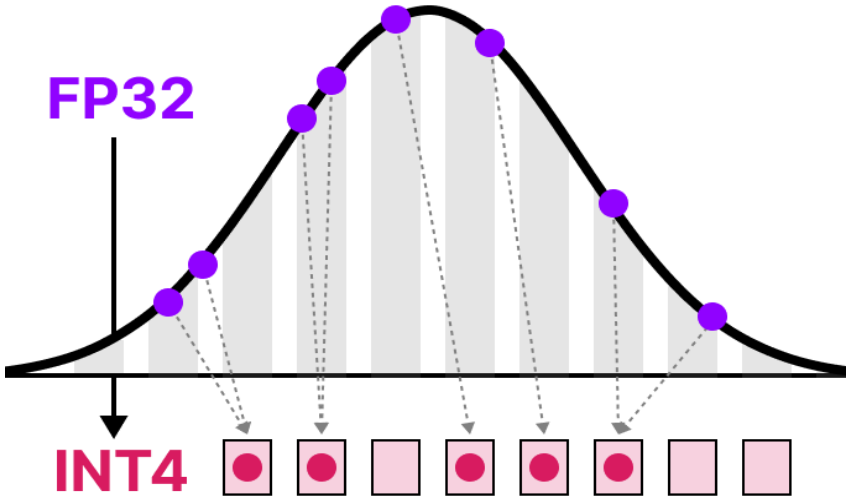

그림 4. 32ibt fp 숫자를 8bit int로 변환

2.2.2. 프롬프트 엔지니어링

Prompt Engineering은 AI 모델로부터 원하는 결과를 효과적으로 얻기 위해 입력 프롬프트를 최적화하는 기술이다. AI의 사고 과정을 유도하고 구조화하는 다양한 기법을 포함한다. 대표적으로 Chain of Thought (COT) Prompting은 AI가 단계별로 추론하도록 유도하여 복잡한 문제 해결 능력을 향상시키며, Zero-shot COT는 예시 없이도 단계적 사고를 유도한다. Self-Consistency Prompting은 AI가 주어진 정보를 바탕으로 여러 번의 추론을 통해 더 일관성 있고 풍부한 결과를 생성하도록 한다. 또한, Role Prompting, Structured Prompting, Instruction Prompting 등의 기법들이 AI의 응답을 더욱 정교하고 목적에 부합하게 만든다. 이러한 프롬프팅 기법들을 적절히 조합하여 사용함으로써, AI 모델의 성능을 극대화하고 더 정확하고 유용한 결과를 얻을 수 있다. 본 연구에서는 위 프롬프트 엔지니어링 기법을 적절히 조합하여 프롬프트를 생성하였다. 일관되고 납득할 만한 결과를 지속적으로 만들어내는 LLM 서버에 적용했다.

2.3. 프론트엔드

● 2.3.1. 역할

프론트엔드는 웹 애플리케이션의 사용자 인터페이스(User Interface)와 사용자 경험(User eXperience)을 담당하는 중요한 역할을 수행하는 부분이다. 프론트엔드는 사용자가 애플리케이션과 상호작용할 수 있게 해주는 모든 시각적 요소를 관리한다. 버튼, 입력 폼, 영화 목록, 추천 영화 등 사용자가 보는 화면과 화면 내의 요소들을 포함하며 이러한 요소들을 통해 사

용자는 애플리케이션의 기능에 쉽게 접근하고 활용할 수 있다.

● 2.3.2. 기술 스택

프론트 엔드 개발은 HTML, CSS, JavaScript와 같은 기본적인 웹 기술로 시작되며, 이를 통해 웹 애플리케이션의 구조, 스타일, 동작을 정의한다. 하지만 복잡한 사용자 인터페이스(UI)를 다루기 위해 최신 웹 프레임워크나 라이브러리가 필요하다. 본 연구에서는 React와 CSS를 활용하여 UI를 구축한다.

<React>

React는 컴포넌트 기반의 라이브러리로, UI를 여러 개의 독립적인 컴포넌트로 나누어 개발한다. 이러한 특성으로 인하여 코드의 재사용성과 유지보수성을 높일 수 있다는 장점이 있다. 또한, React는 가상 DOM(Document Object Model)을 활용하여 효율적인 업데이트를 지원, 애플리케이션의 성능을 최적화할 수 있다.

<CSS>

UI를 스타일링하는 데 사용되며, 웹 페이지의 레이아웃과 시각적 스타일을 결정한다. 본 연구에서는 CSS 모듈 및 Styled Components를 활용하며, 이를 통해 각 컴포넌트의 스타일을 개별적으로 관리한다.

2.4 백엔드

백엔드는 웹 애플리케이션의 서버 측으로, 클라이언트(프론트엔드)에서 요청한 데이터를 처리하고 응답을 반환하는 역할을 한다. 백엔드는 데이터베이스와 연결되어, 애플리케이션 로직을 실행하고, 클라이언트의 요청에 따라 데이터를 처리하여 반환한다. 본 연구에서 백엔드는 Flask와 MySQL을 사용해 구축된다.

<Flask>

Flask는 Python 기반의 마이크로 웹 프레임워크로, 웹 서버를 구축하고 RESTful API를 작성하는 데에 강점을 가지고 있다. 또한 Flask는 간단하고 유연한 구조로 빠르게 API 서버를 구축을 가능케 한다. REST API 엔드포인트를 정의하여 클라이언트로부터 요청을 받고, 데이터베이스에서 필요한 정보를 처리한 후 JSON 형식으로 응답을 반환한다.

<MySQL>

MySQL은 데이터를 테이블(Relational Table)로 저장하며, SQL 쿼리를 통해 데이터를 조회, 삽입, 수정, 삭제하는 작업을 수행한다. 사용자의 영화 선호도, 시청 기록, 추천 결과 등을 테이블 형식으로 관리하며, SQL 쿼리를 통해 데이터베이스에서 필요한 정보를 조회하거나 갱신한다.

2.5 모델 서버

<FastAPI>

FastAPI는 현대적이고 빠른 웹 API를 구축하기 위한 파이썬 프레임워크다. 자동 API 문서 생성, 타입 힌팅 기반의 데이터 검증, 비동기 처리 지원, 높은 성능이 특징이며, Pydantic을 통한 데이터 모델링과 검증을 제공한다. 이 실험에서는 Pydantic 모델을 통해 데이터의 유효성을 검증하고, 에러 처리를 하며 llm추론 API엔드 포인트를 제공하는 웹 서버 역할을 한다.

<LangChain>

LangChain은 LLM(Large Language Model) 애플리케이션 개발을 위한 프레임워크로, 프롬프트 관리, 체인 구성, 메모리 관리, 에이전트 생성 등 다양한 기능을 제공한다. 특히 복잡한 LLM 워크플로우를 체계적으로 구성할 수 있게 해주며, 템플릿 기반의 프롬프트 관리와 체인을 통한 순차적 처리를 지원한다. 이 실험에서 구조화된 프롬프트 생성 및 LLM응답 처리를 담당하여 일관된 응답을 생성하도록 구성했다.

3. 연구 내용

3.1. 구성원 별 역할

이름	역할 분담
박지환	● 데이터 분석 및 전처리 ● 추천 시스템 구축 ● LLM 모델 선정 및 테스트(EEVE-KR-Instruct) ● 시스템 아키텍처 설계
장재혁	● API 설계 및 구현

	● 결과 시각화 ● 백엔드 개발
하현진	● 웹 ui 구성 및 프론트엔드 개발(react) ● 프롬프트 엔지니어링 ● 데이터 전처리 • 모델 서빙

표 1. 구성원 별 상세 역할

○

3.2. 데이터 확보 및 전처리

3.2.1. 데이터 확보

연구 진행 초기에 Kaggle의 The Movies 데이터셋을 활용한 테스트를 진행한 바가 있다. 테스트 결과 해당 데이터셋은 특성의 종류는 충분히 갖추고 있으나 7년 전까지의 영화 데이터만을 포함하고 있기 때문에 현재 사용자들에게 추천을 제공하기에는 정보가 매우 노후화되었고, 최신 데이터의 공백을 보완하는 작업을 LLM에 일임하기에는 무리가 있다고 판단하여 연구 팀 논의 후 데이터셋 교체를 결정하였다.

여러 종류의 데이터셋을 비교한 결과, kaggle의 Movies Daily Update 데이터셋을 선정하여 전처리 후 실제 시스템에 적용하였다. 해당 데이터셋의 세부 사항은 3.3.2절에서 후술한다.

3.2.2. 데이터 분석

본 데이터셋은 단일 csv 파일로, 총 20개의 열로 구성되어 있다. 고유 id, 제목, 평점 수 및 평균, 예산, 언어 등 여러 종류의 특성들을 포함하고 있는데, 그 중 시스템에서 주요하게 활용되는 열들은 아래와 같다.

- id(int) : 각 영화에 고유하게 부여된 식별 번호
- title(str) : 영화의 제목
- credits(str) : 영화의 출연진 정보. 하이픈(-)으로 구분되어 있다.
- genres(str) : 영화의 장르 정보. 단일 값이 아니며, 하이픈(-)으로 구분되어 있다.
- keywords(str) : 영화를 표현할 수 있는 키워드. 하이픈(-)으로 구분되어 있다.
- vote_average(float) : TMDB(The Movie DataBase) 사이트 내 평점 평균값
- vote_count(float) : TMDB(The Movie DataBase) 사이트 내 평가 수

3.2.3. 데이터 전처리

<CSV 파일 전처리 과정>

우선 데이터셋 내 주요하게 활용되는 열 중 genres 및 credits에서 NULL 값을 갖는 행들은 후속 과정에서 이루어질 추천 과정 자체에 적용되기에는 어려움이 있다고 판단하여 삭제 처리를 진행하였다. 데이터셋 정보 요약을 참고하여 genres의 경우 약 21만 개의 행이, credits의 경우 약 22.5만 개의 행이 이에 해당되었음을 확인하였다.

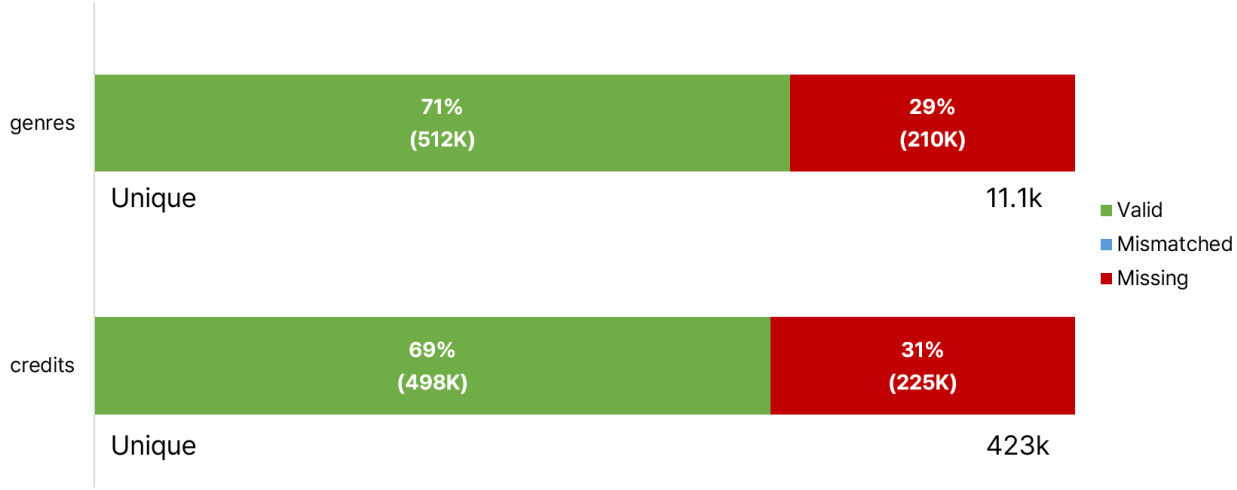

차트 1. 데이터셋 내 genres 및 credits 가용 데이터 비중

뒤이어 release_date 열에서 앞 네자리 만을 추출하여 별개의 release_year 열로 저장하였고 genres, credits, production_companies, keywords 네 개의 열을 리스트 형태로 활용하기 위하여 하이픈(-)을 기준으로 split하여 리스트화하였다.

추가적으로 미개봉 영화 및 신뢰성이 낮은 영화들을 제외하기 위하여 평점 평균과 평가 수 각 상위 50%를 기준으로 하여 유효 기준을 연산하고 점검하는 과정을 진행하였다.

```
vote_counts = meta[meta['vote_count'].notnull()]['vote_count'].astype('int')
vote_averages = meta[meta['vote_average'].notnull()]['vote_average'].astype('float')
vote_counts_minimum = vote_counts.quantile(0.5)

print("유효한 평점 평균들의 평균값 : {}점".format(vote_averages.mean()))
print("유효한 영화(평가 수 상위 50% 이내) 판별 기준 평점 수 : {}개".format(vote_counts_minimum))
Executed at 2024.10.18 02:07:15 in 93ms

 유효한 평점 평균들의 평균값 : 2.3375483499414402점
 유효한 영화(평가 수 상위 50% 이내) 판별 기준 평점 수 : 0.0개
```

유효 기준 연산 결과, 유효한 평점 평균들의 평균값은 약 2.34점으로, 유효한 평점 수는 0.0개로 계산되었는데, 최신 영화의 경우 최신화된 데이터셋을 사용한다 하더라도 평점 수가

아직 없는 경우가 있음을 확인한 바 있다. 이에 따라 vote_average 열에서 결측치(NULL)를 갖는 열만 삭제 처리한 뒤 후속 과정을 진행하였다.

최종적으로 추천 및 결과 출력에 활용될 열들만(id, title, genres, release_year, credits, production_companies, vote_average, vote_count, keywords, poster_path) 별개로 추출하여 각 열의 값에 맞는 자료형(int, float, str)으로 캐스팅 처리한 후 별도의 파일(preprocessed_new.csv)로 저장하였다.

해당 파일에 포함된 영화(행)의 수는 총 276,010개로 확인되었다.

<다중 레이블 특성 벡터화>

다중 레이블을 갖는 특성들(genres, credits, keywords)을 추천 시스템에 활용하기 위하여 벡터화 과정을 진행하였다. 해당 과정에서 scikit-learn 라이브러리의 preprocessing 패키지가 사용되었다.

각 열마다 처리를 수행할 MultiLabelBinarizer 객체를 선언하였고, 메모리 효율성을 위하여 그 출력 형태로 희소 행렬을 선택하였다(sparse_output=True). 이후 데이터셋의 각 열에 맞게 인코딩을 진행하였고, 인코딩된 각 희소 행렬들을 결합(hstack)하는 작업에 이어 원본 정보와 인코딩된 특성을 한 데이터 프레임에 갖도록 한 번 더 결합하였다. 대규모 데이터셋 내 다중 레이블 특성들을 벡터화하되, 그 형태는 희소 행렬로 하고 원본 데이터와 결합함으로써 메모리 자원 요구량을 줄일 수 있도록 진행하였다.

```
from scipy.sparse import hstack
# 각 컬럼 별 mlb 선언, 희소 행렬 출력 설정
mlb_genre = MultiLabelBinarizer(sparse_output=True)
mlb_actor = MultiLabelBinarizer(sparse_output=True)
mlb_keyword = MultiLabelBinarizer(sparse_output=True)

genre_encoded = mlb_genre.fit_transform(data['genres'])
actor_encoded = mlb_actor.fit_transform(data['credits'])
keyword_encoded = mlb_keyword.fit_transform(data['keywords'])

# 희소 행렬 사용, 데이터 결합
encoded_features = hstack([genre_encoded, actor_encoded, keyword_encoded])

# 원본 데이터(data)와 인코딩 된 특성을 결합
data_encoded = pd.concat([
    data,
    pd.DataFrame.sparse.from_spmatrix(
        encoded_features,
        columns=mlb_genre.classes_.tolist() + mlb_actor.classes_.tolist() + mlb_keyword.classes_.tolist(),
        index=data.index
    )
], axis=1)
```

최종적으로 scipy 라이브러리의 sparse 패키지, joblib 모듈을 사용하여 본 과정에서 생성된

희소 행렬 및 인코딩 특성이 없는 원본 데이터셋, 그리고 MultiLabelBinarizer 객체와 인코딩된 특성 이름까지 각각 파일로 저장하여 일관성 유지를 통한 변수 차단 및 추후 추천 시스템 단계에서의 활용이 가능하도록 하였다.

<고유 키워드 저장>

일반적인 구술 문장 형태의 사용자 선호 키워드가 LLM으로 전달되었을 때 일반적인 오탈자 범위에서 원본 데이터셋의 키워드와 어긋나는 현상을 방지하기 위하여 데이터셋 내 특성들의 고유 키워드를 전체 추출하는 과정을 진행하였다.

각 열의 아이템 값을 리스트로 변환한 뒤, 'name'에 해당되는 부분만 추출하여 중복 없이 별도의 리스트에 저장하여 return하는 extract_unique_items 함수를 구현하였고 이를 genres, credits, keywords 세 종류의 열에 적용하여 고유 키워드를 추출해 내었다.

```
def extract_unique_items(column):
    unique_items = set()
    for item in column:
        try:
            # 문자열을 리스트로 변환
            item_list = ast.literal_eval(item)
            # 리스트의 각 항목에서 이름만 추출
            for sub_item in item_list:
                if isinstance(sub_item, dict) and 'name' in sub_item:
                    unique_items.add(sub_item['name'])
                elif isinstance(sub_item, str):
                    unique_items.add(sub_item)
        except:
            pass
    return sorted(list(unique_items))


# genres, credits(actors), keywords에서 고유한 항목 추출
unique_genres = extract_unique_items(data['genres'])
unique_actors = extract_unique_items(data['credits'])
unique_keywords = extract_unique_items(data['keywords'])
```

마지막으로, 추출된 리스트 결과를 각각의 json 파일에 utf-8 인코딩을 통해 저장하여 LLM이 참조할 수 있는 형태로 생성하였다.

```
# 결과를 JSON 파일로 저장
with open('unique_genres.json', 'w', encoding='utf-8') as f:
    json.dump(unique_genres, f, ensure_ascii=False, indent=4)

with open('unique_actors.json', 'w', encoding='utf-8') as f:
    json.dump(unique_actors, f, ensure_ascii=False, indent=4)

with open('unique_keywords.json', 'w', encoding='utf-8') as f:
    json.dump(unique_keywords, f, ensure_ascii=False, indent=4)
```

3.3. 추천 시스템

3.3.1. 시행착오

<LightFM 라이브러리 적용 시도>

최초 계획에서 언급된 하이브리드 추천 시스템 라이브러리인 LightFM을 추천 시스템에 적용하려는 시도가 있었다. 하지만 라이브러리 사용 부분에서 아나콘다를 활용하여 별도의 가상환경으로 패키지 버전을 제어하였으나 Numpy, Scipy 등의 여러 패키지들의 버전이 충돌하여 호환성 관련 문제가 발생하였고, 적절한 수정을 통하여 기존 데이터셋 필요 형식에 맞추어 변형한 입력 데이터셋을 기반으로 LightFM 모델을 구축하고 학습하는 데에는 성공하였으나 최종 출력값이 'c', 'a' 와 같은 단일 알파벳이나 특수 문자로 출력되는 현상이 발생하여 수정을 시도하였다.

그러나 매 개선 후 모델을 새롭게 구축하고 학습하는 과정에서 GPU 사용 환경 내에서 진행되었음에도 불구하고 소요 시간이 매우 길게 요구되는 문제가 확인되었다. 추가적으로 배치 크기 및 학습률 설정 등의 작업을 수행하여 소요 시간 단축을 시도하였으나 명확한 개선 결과가 확인되지 않았고, 데이터셋을 축소하는 과정에서 수반되는 유효 데이터 삭제의 위험성 및 시간적 제약 사항을 고려하여 해당 계획을 파기하고 후술할 계획으로 변경하여 진행하였다.

<신경망 적용 시도>

Tensorflow 라이브러리를 활용하여 추천 수행 신경망 구축을 시도하였고, 테스트 단계에서 실험되었던 신경망의 구조는 아래와 같다.

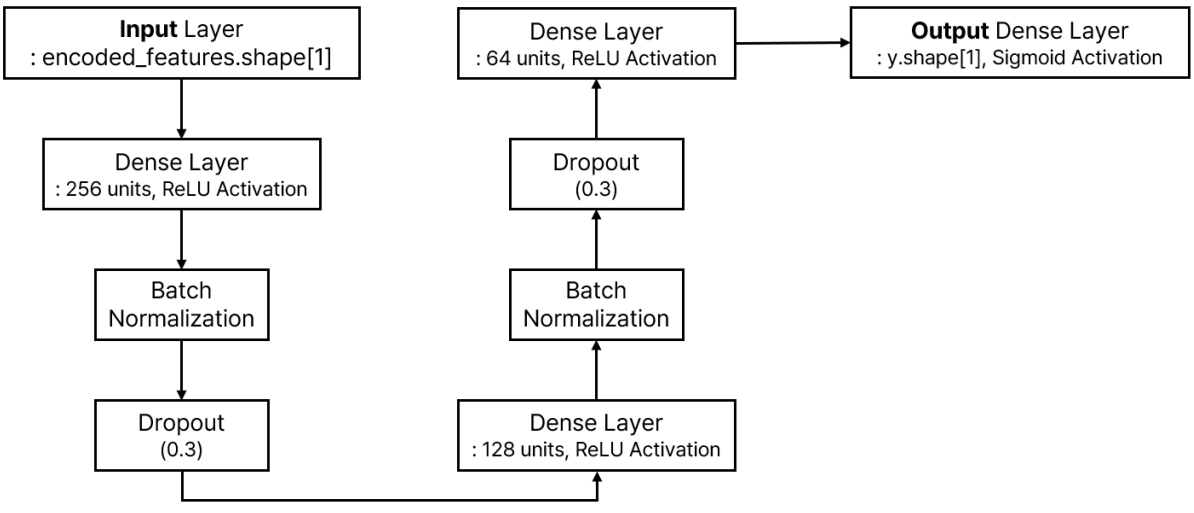

그림 5. 테스트용 신경망의 구조

학습 및 추론은 RTX 3060 GPU를 기반으로 한 CUDA 환경 상에서 수행되었다. 전처리가 완료된 아이템 특성 데이터셋을 기반으로 학습률 스케쥴링 및 조기 종료 기능을 적용한 30

epoch 학습을 진행하였다. 이후 데이터셋 및 이전 단계에서 저장한 MLB 파일을 활용한 추천 성능 테스트를 진행하였는데 주어진 장르와 배우, 키워드의 반영은 충실히 이루어졌으나, GPU 사용 환경 내에서 구동되었음에도 불구하고 추천 산출까지의 평균 소요 시간이 2분 12초로 매우 길게 측정되었다. 이에 따라 별도의 후속 경량화 조치가 필요하다고 판단되어 계획 변경 절차를 진행하였다.

3.3.2. 하이브리드 추천 시스템

사용자의 선호 관련 키워드를 받아 후보 세트를 구성하는 데에만 추천 시스템이 사용될 것이기 때문에, 연산 자원 부담 및 시간 요구량을 최소화하기 위하여 코사인 유사도를 활용한 추천 시스템을 구축하였다. 데이터셋 내 영화의 특성과 사용자 선호 키워드의 유사도를 측정한 뒤 평가 관련 정보를 활용하여 생성한 신뢰도 점수와 함께 가중 선형 결합(Weighted Linear Combination)을 수행하도록 구현하였다.

테스트 단계에서는 장르, 배우, 키워드를 4:4:2의 비율로 가중치를 적용하여 추천을 수행하도록 하였는데, 이러한 방식의 경우 평점이 좋지 못한 영화 또는 개봉한 지 얼마 되지 않아 평가 정보가 부족하거나 없는 영화('조커 : 폴리 아 되' 등의 예)가 추천에 반영되지 않는 문제가 발견되어 추가적인 개선 작업을 진행하였다.

<필요 파일 로드>

데이터 전처리 과정에서 별도로 저장해 둔 MLB 파일과 인코딩된 특성, 추천 결과에 추가 정보를 작성하는 과정에서 참조하기 위한 데이터셋을 로드하여 준비하였다.

<신뢰도 점수 연산>

극단적인 평점의 영향을 줄여 추천 결과가 편향되는 현상 및 투표 수가 적은 영화가 과대평가되는 현상을 방지하고 투표 수가 많은 영화의 신뢰도를 보정하기 위하여 베이지안 평균(Bayesian Average)를 기반으로 한 신뢰도 점수 연산을 적용하였다.
평가가 단 하나도 존재하지 않는 영화의 경우 전체 영화 평점의 평균으로 점수를 부여받고, 평가 수가 하나라도 있는 영화의 경우에는 아래 식에 따라 신뢰도 점수가 연산되게 된다.

$$\text{Reliability\_Score} = \frac{\text{vote\_count}}{\text{vote\_count} + \text{min\_votes}} \cdot \text{vote\_average} + \frac{\text{min\_votes}}{\text{vote\_count} + \text{min\_votes}} \cdot \text{mean\_rating}$$

min_votes는 베이지안 평균에서의 사전 가중치처럼 작동하는데, 이는 투표 수가 적은 영화의 경우 극단적인 평점에 크게 영향을 받지 않고 전체 평균 평점과 유사한 수준의 영향을 받도록 조정하기 위하여 설정하였다.

위 식을 적용함에 따라서 투표 수가 많은 영화는 실제로 평가된 vote_average 값에 근접하도

록 가중치를 부여받고, 반대로 투표수가 적은 영화의 경우 mean_rating 값에 근접하도록 보정하였다.

<가중 선형 결합 및 유사도 계산>

유사도 계산에서는 배우, 장르, 키워드, 평가 수, 평점 평균, 신뢰도 점수가 아래와 같이 배정되어 적용되었다.

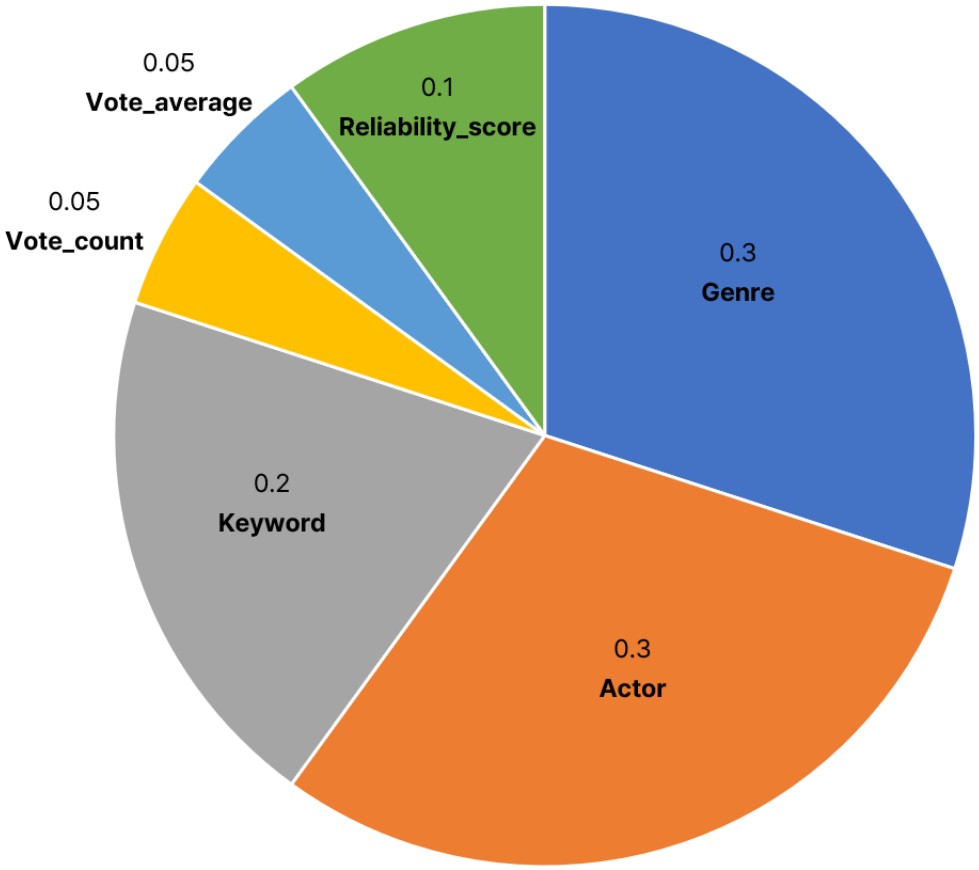

차트 2. 추천 시스템에서의 특성 별 가중치 배정 비율

MLB를 각 특성 벡터에 맞게 변형한 뒤 각 특성별 종료 인덱스를 주어진 벡터의 shape를 활용하여 구하였고, 이를 기반으로 각각의 가중치를 곱하여 genres, actors, keywords에 대한 가중치 적용을 진행하였다.

```
input_vector = csr_matrix((1, encoded_features.shape[1]))

genre_end = genre_vector.shape[1]
actor_end = genre_end + actor_vector.shape[1]
keyword_end = actor_end + keyword_vector.shape[1]

input_vector[:, :genre_end] = genre_vector * genre_weight
input_vector[:, genre_end:actor_end] = actor_vector * actor_weight
input_vector[:, actor_end:keyword_end] = keyword_vector * keyword_weight

# 유사도 계산
similarities = encoded_features.dot(input_vector.T).toarray().flatten()
```

평점 평균, 평가 수 및 신뢰도 점수는 위 세 특성과 별개로 정규화 과정을 진행하였는데 이는 최소 값과 현재 값의 편차를 최대 값과 최소 값의 편차로 나누어 진행하였고, 이를 수식으로 나타내면 아래와 같다.

$$\text{normalized\_feature} = \frac{\text{feature\_value} - \text{minimum\_value}}{\text{maximum\_value} - \text{minimum\_value}}$$

가중 처리된 세 특성을 활용한 유사도와 정규화 처리된 세 특성을 가중 처리하여 병합한 뒤, NumPy 라이브러리의 argsort 함수를 활용하여 top_n개 까지의 영화 정보를 정리하여 출력한다.

```
선호하는 장르를 쉼표로 구분하여 입력해주세요 (예: Action, Adventure): Action
선호하는 배우를 쉼표로 구분하여 입력해주세요 (예: Tom Cruise, Brad Pitt): Keanu Reeves
선호하는 영화 키워드를 쉼표로 구분하여 입력해주세요 (예: superhero, time travel): killer
```

그림 6. 추천 시스템 테스트 입력부

```
추천 영화:
- The Matrix
  장르: ['Action', 'Science Fiction']
  출연: ['Keanu Reeves', 'Laurence Fishburne', 'Carrie']
  키워드: ['man vs machine', 'martial arts', 'dream', 'artificial intelligence (a.i.)', 'saving the world',

- John Wick
  장르: ['Action', 'Thriller']
  출연: ['Keanu Reeves', 'Michael Nyqvist', 'Ian McShane']
  키워드: ['hitman', 'gangster', 'secret organization', 'bratva (russian mafia)', 'revenge', 'murder', 'dog'

- John Wick: Chapter 3 - Parabellum
  장르: ['Action', 'Thriller', 'Crime']
  출연: ['Keanu Reeves', 'Halle Berry', 'Ian McShane']
  키워드: ['new york city', 'martial arts', 'bratva (russian mafia)', 'casablanca morocco', 'secret society'
```

그림 7. 해당 출력에 따른 추천 값의 일부

위의 이미지는 John Wick 시리즈를 출력값으로 의도하고 진행되었던 한 테스트의 입력 및

결과값이다. 가중치 파라미터 중에서 선호 장르(Action)와 배우(Keanu Reeves)의 값이 가장 크게 설정되어 있는데, 그 영향으로 추천 영화 목록의 1순위로는 의도와는 다소 다른 결과(The Matrix)가 출력됨을 확인하였다. 하지만 그 이후의 추천 결과에는 본 의도대로 적절한 영화들이 선정되어 출력되었음을 확인하였고, 이후 LLM의 추가적인 추론 과정에 제시하기에 큰 문제가 없음을 여러 차례의 테스트를 통하여 확인하였다.

3.4. 대형 언어 모델

3.4.1. 모델 서버 및 프롬프트

main.py, organize.py, perona.py, recommendation.py, utils.py 파일로 구분하여 모듈화 작업을 진행하였고, llm이 적절한 답변을 생성하도록 앞서 2.2.2 프롬프트 엔지니어링에 서술된 기법을 이용해 각 작업에 대한 프롬프트를 작성했다.

<main.py>

사용자의 입력을 받아 각 유저의 페르소나를 생성한다. 나이, 성별, 직업을 기본으로 하여 입력으로 제공된 추가 정보를 이용해 영화 추천 사용할 수 있는 적절한 페르소나를 생성하여 리턴한다. 또한 추가적인 입력이 들어왔을 때 페르소나를 업데이트 할 수 있는 기능을 수행한다.

<persona.py>

유저의 페르소나와 영화 시청 기록, 영화 선호도, 영화 시청 환경, 시간등을 고려해 최종적인 영화 추천을 진행한다. 추천 시스템에서 선정된 영화 후보군과 llm이 적절하다고 판단하는 영화를 추가로 제시해 최종 추천 영화를 선정한다. diversity와 Novelty를 확보하도록 프롬프트를 구성하여 단순한 추천 시스템을 통해 적절한 영화 추천을 할 수 있도록 유도했다. 해당 파일에서 유저 페르소나를 리턴하여 DB에 저장한다.

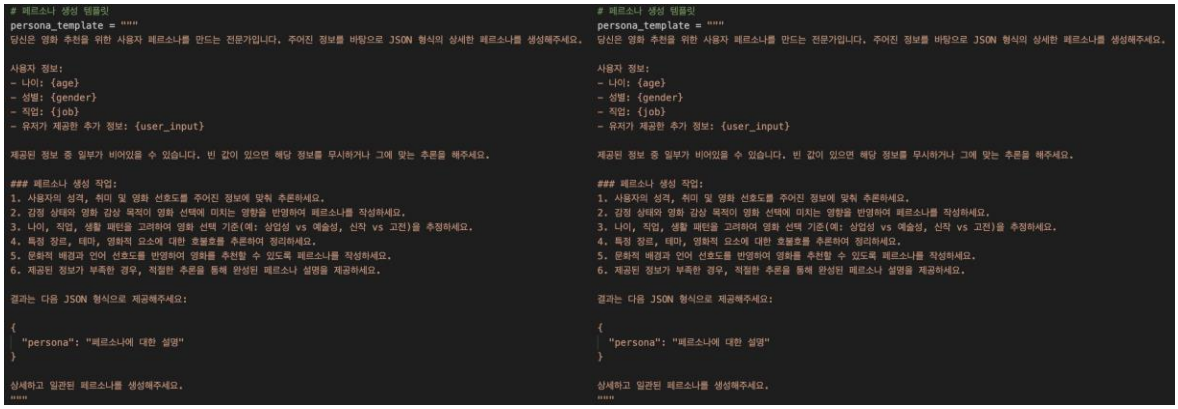

그림 8. llm 서버에 사용된 프롬프트 중 일부

<organize.py, utils.py>

각각 입출력을 구조화하기 위한 모듈이다.organize.py는 사용자가 추가로 입력한 정렬되지않은 무작위 입력을 정리하여 리턴한다. 정리된 정보는 페르소나 업데이트 작업에 사용된다. utils.py는 JSON 형식의 문자열을 안전하게 파싱하기 위한 두 개의 함수를 정의하여 입력 텍스트에서 JSON 문자열을 추출하고 불필요한 공백을 제거하여 전처리하며, 전처리된 텍스트를 JSON 객체로 변환하려고 시도한다. 파싱에 성공하면 JSON 객체를 반환하고, 실패할 경우 오류 정보를 포함한 딕셔너리를 반환하여 JSON 데이터 처리 과정에서 발생할 수 있는 문제를 관리한다.

3.5. 프론트 엔드

3.5.1. 프로필 업데이트

3.5.1.1. 기능 설명

프로필 업데이트 기능은 사용자가 자신의 나이, 성별, 이름, 직업을 입력하고 이를 서버로 전송하여 업데이트하는 기능이다. 이 정보를 통해 개인화된 영화 추천을 제공할 수 있다.

3.5.1.2. 구현 방식

<다이얼로그와 입력 필드>

Dialog 컴포넌트를 사용하여 프로필 업데이트 폼을 다이얼로그로 구성하였다. 해당 업데이트 폼에는 나이, 성별, 이름, 직업 입력 필드가 포함되어 있다. 사용자가 각 필드를 입력할 경우, onChange 이벤트 핸들러가 호출되어 useState를 통하여 상태를 관리한다.

<서버로의 데이터 전송>

사용자가 [프로필 업데이트] 버튼을 클릭하면 handleUserInfoUpdate 함수가 호출된다. 이 함수는 입력된 정보를 서버로 PUT 요청을 통해 전송하는 기능을 수행한다. 이렇게 요청을 전송한 뒤, 서버에서 업데이트가 성공할 경우 사용자는 '프로필 업데이트 성공' 메시지를 받는다.

3.5.2. 영화 취향 공유

3.5.2.1. 기능 설명

사용자는 선호하는 영화 장르와 배우를 입력하여 자신의 영화 취향을 공유할 수 있다. 또한, 해당 정보를 바탕으로 더욱 개인화된 영화 추천을 받을 수 있게 된다.

3.5.2.2. 구현 방식

<장르 및 배우 입력>
사용자는 Input 컴포넌트를 통해 장르와 배우를 입력한다. 이렇게 입력된 값은 각각 handleGenreSearchChange와 handleActorSearchChange 함수를 통해 실시간 검색으로 처리된다. 해당 과정을 통하여 매칭되는 영화 장르와 배우를 Dropdown 리스트로 출력한다.

<서버로 취향 공유>
사용자가 입력을 완료한 후, [취향 공유하기] 버튼을 클릭하면 updateUserPreferences 함수가 호출되고, 이 함수는 서버로 POST 요청을 전송, 이를 통하여 영화 취향 데이터를 전송한다.

3.5.3. 본 영화 기록

3.5.3.1. 기능 설명

사용자는 자신이 본 영화 제목을 입력하고, 이를 시청 기록으로 저장할 수 있다. 또한 입력한 영화 제목을 통해 관련 영화를 추천받을 수 있다.

3.5.3.2. 구현 방식

<영화 제목 입력>

Input 컴포넌트를 사용하여 영화 제목을 입력하고, handleSearchChange 함수로를 활용하여 실시간 검색을 처리한다.

<시청 기록 표시>

사용자가 추천된 영화를 클릭하거나 검색 결과에서 선택하면, 해당 영화가 시청 기록에 추가된다. 시청 기록은 watchedMovies 배열에 저장되어 화면에 출력된다.

3.5.4. 자유 서술 입력

3.5.4.1. 기능 설명

사용자는 자신의 영화 취향이나 생각을 자유롭게 서술할 수 있다. 해당 서술 내용을 활용하여 추천 시스템에 개인적 선호에 관한 정보들을 제공할 수 있다.

3.5.4.2. 구현 방식

<서술 입력 필드>

사용자가 자유롭게 서술할 수 있도록 Input 컴포넌트를 사용하여 텍스트 입력을 받는다. 해당 입력값은 description 상태에 저장된다.

<서버로 제출>

서술 내용을 작성 완료하였을 경우, handleSubmit 함수가 호출되어 서버로 POST 요청을 전송, 이를 통하여 서술 내용을 전송한다.

3.5.5. 새로운 영화 발견하기

3.5.5.1. 기능 설명

사용자는 버튼을 클릭하여 추천 영화 목록을 추가로 로드할 수 있다. 해당 기능은 새로운 영화를 발견하고, 추천 목록을 확장하는데 도움을 줄 목적으로 구현되었다.

3.5.5.2. 구현 방식

<버튼 클릭 시>

사용자가 "새로운 영화 발견하기" 버튼을 클릭할 경우, fetchAdditionalMovies 함수가 호출

되어 서버로부터 새로운 영화 목록을 가져온다. 이렇게 가져온 영화 목록은 배열에 추가되어 사용자에게 더 많은 영화를 추천할 수 있게 한다.

3.6. 서비스 구조 설계

전반적인 시스템의 서비스 작동 구조를 나타낸 흐름도는 아래와 같다.

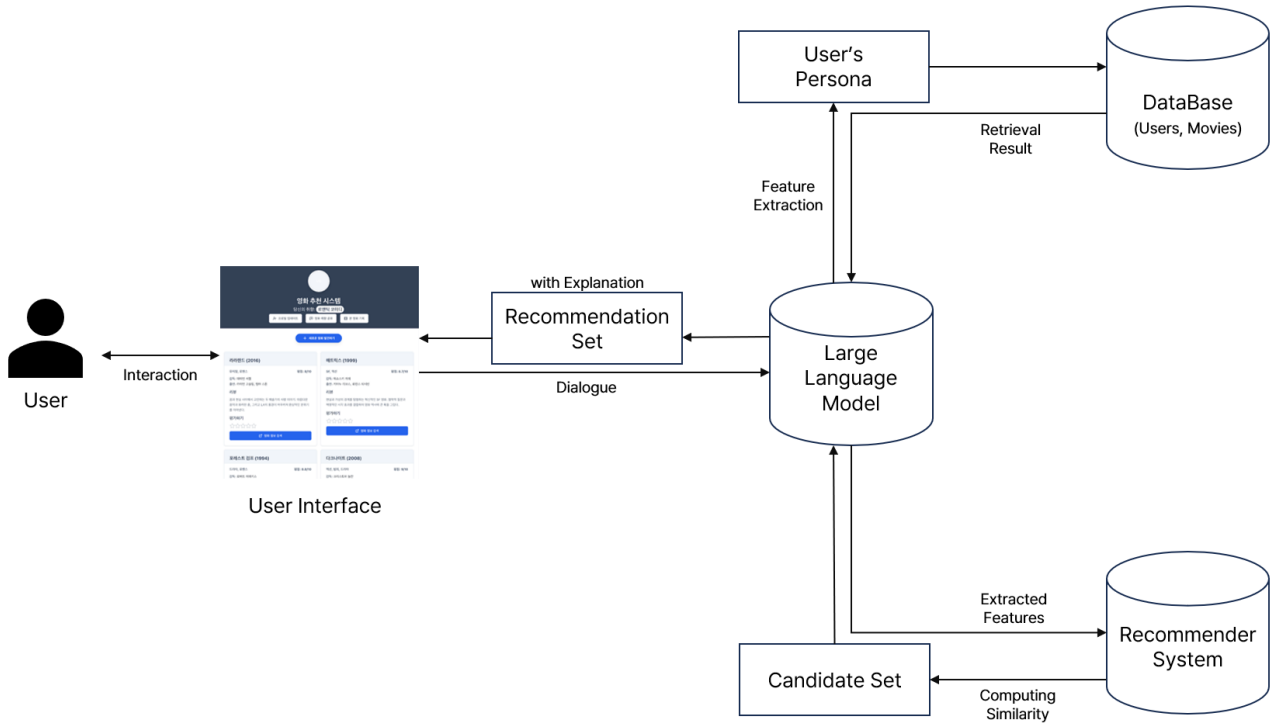

그림 9. LLecommend 전체 서비스 흐름도

3.6.1. 요소 설명

3.6.1.1. User

서비스의 기능 제공 대상이 되는 객체로, user interface를 통하여 본인의 선호 정보를 자유롭게 입력하고, 추천 결과를 제공받아 평가한다.

3.6.1.2. User Interface

User와 LLM 사이의 정보 교환의 매개 역할을 수행한다. 사용자의 프로필 입력 및 편집, 추천 결과 및 근거 출력을 웹의 형태로 구현한다.

3.6.1.3. Large Language Model(LLM)

EEVE-Korean-Instruct-10.8B-v1.0 모델이 사용되었으며, 수행하는 기능은 아래와 같다.

- 사용자의 발화 내용 내 선호 관련 특성 키워드 추출

- 추출된 특성 키워드들을 규격화하여 recommender system으로 전달
- 키워드를 활용한 user's persona 생성
- Database 참조를 통한 영화 및 프로파일 정보 검색
- Candidate set과 user's persona를 기반으로 한 추론, Recommendation set 생성
- Recommendation set의 각 아이템들에 대한 추천 근거 생성

3.6.1.4. User's Persona

User가 user interface를 통하여 자유롭게 입력한 본인의 선호 정보가 프롬프트에 적용될 수 있도록 LLM을 통해 가공된 형태이다. user의 성별, 나이, 선호 장르, 배우 및 기타 선호 특성 정보들이 일반 문장 형태로 표현된다.

3.6.1.5. Database

User's persona 및 영화의 자세한 정보를 갖는 원본 데이터셋을 저장한다. LLM에서 추가적으로 검색이 필요한 정보가 발생할 경우, database 내의 정보를 검색하는 과정이 이루어지게 된다.

3.6.1.6. Recommender System

가중 선형 결합 및 코사인 유사도를 기반으로 추천 점수를 산출하는 시스템으로, LLM에 의하여 추출된 사용자 선호 키워드를 입력으로 전달받아 유사성을 계산, 상위의 영화들을 추출하여 candidate set을 생성한다.

3.6.1.7. Candidate Set

Recommender system의 출력으로 생성된 후보 아이템 세트이다. 원본 데이터셋을 매 번 참조할 경우 연산 부담이 과중해지는 현상을 방지하기 위해 고안되었고, 이는 LLM의 입력으로 전달되어 최종 추천 결과인 recommendation set을 생성하는 기반이 된다.

3.6.1.8. Recommendation Set

Candidate set을 입력 받은 LLM이 최종적으로 출력하는 추천 결과 세트이다. 해당 세트 내의 정보들은 LLM에 의해 한 번 더 가공되어 user interface를 통하여 user에게 출력된다.

3.6.2. 작동 흐름

작동 흐름은 아래의 순서와 같다.

1. 사용자가 웹 페이지에 접속, 사용자 프로필을 생성한다. 나이, 성명, 직업 세 종류의 정보만을 입력하고, 입력된 정보는 사용자 고유 식별을 위한 id값과 함께 users 테이블에 저장된다.
2. 사용자가 자유 발화 탭을 열어 영화 추천에 필요한 정보를 포함한 구술형 문장을 자유롭게 입력한다. 해당 문장의 양식에 별도의 제한은 없고, 작성된 문장은 LLM의 입력으로 전달된다.
3. LLM에서 전달된 문장을 받아 영화 추천에 유용한 키워드를 추출한다. 선호 배우, 장르, 취미, 나이 등의 정보 키워드들을 추출하고 이를 활용하여 프롬프트에 따라 각 사용자의 user's persona를 생성, users 테이블 내 현재 사용자에 해당하는 행의 persona 열의 값으로 저장한다.
4. 사용자가 영화 추천을 요청할 시, LLM에서 추출된 주요 내용 중 선호 배우 및 장르, 키워드는 규격화되어 추천 시스템으로 전송된다.
5. 추천 시스템에서는 전송된 정보들을 입력으로 받아 유사도 계산을 수행, 이를 기반으로 하여 candidate set을 생성하고 LLM의 입력으로 재전송한다.
6. 전송된 candidate set을 입력으로 받은 LLM이 user's persona와 candidate set의 정보를 기반으로 하여 최종 recommendation set을 구성한다. 또한 LLM은 추천할 영화의 세부 사항을 참조하기 위하여 데이터셋을 참조하거나 예외사항이 발생할 경우 추론 과정을 진행한다. 그 후, recommendation set 내의 각 영화에 대하여 추천 근거 관련 설명을 생성한다.
7. LLM에서 생성된 정보들을 웹으로 다시 출력하고, 그 형태는 후술할 테스트 화면과 같다.
8. 사용자는 추천을 재요청하거나 자유 발화 내용을 추가하여 user's persona를 수정할 수 있는데, 해당 사항의 경우 변동된 프롬프트 내용을 반영하여 각 5번, 3번 단계로 돌아간다.

4. 연구 결과 분석 및 평가

4.1. 연구 결과 분석

구축한 시스템에 여러 가지 사용자 프로파일을 입력하고 테스트를 진행하였다. 그 중 하나의 예시를 들어 결과를 분석하고 평가를 진행한다.

<사용자 프로필 생성 및 자유 발화>

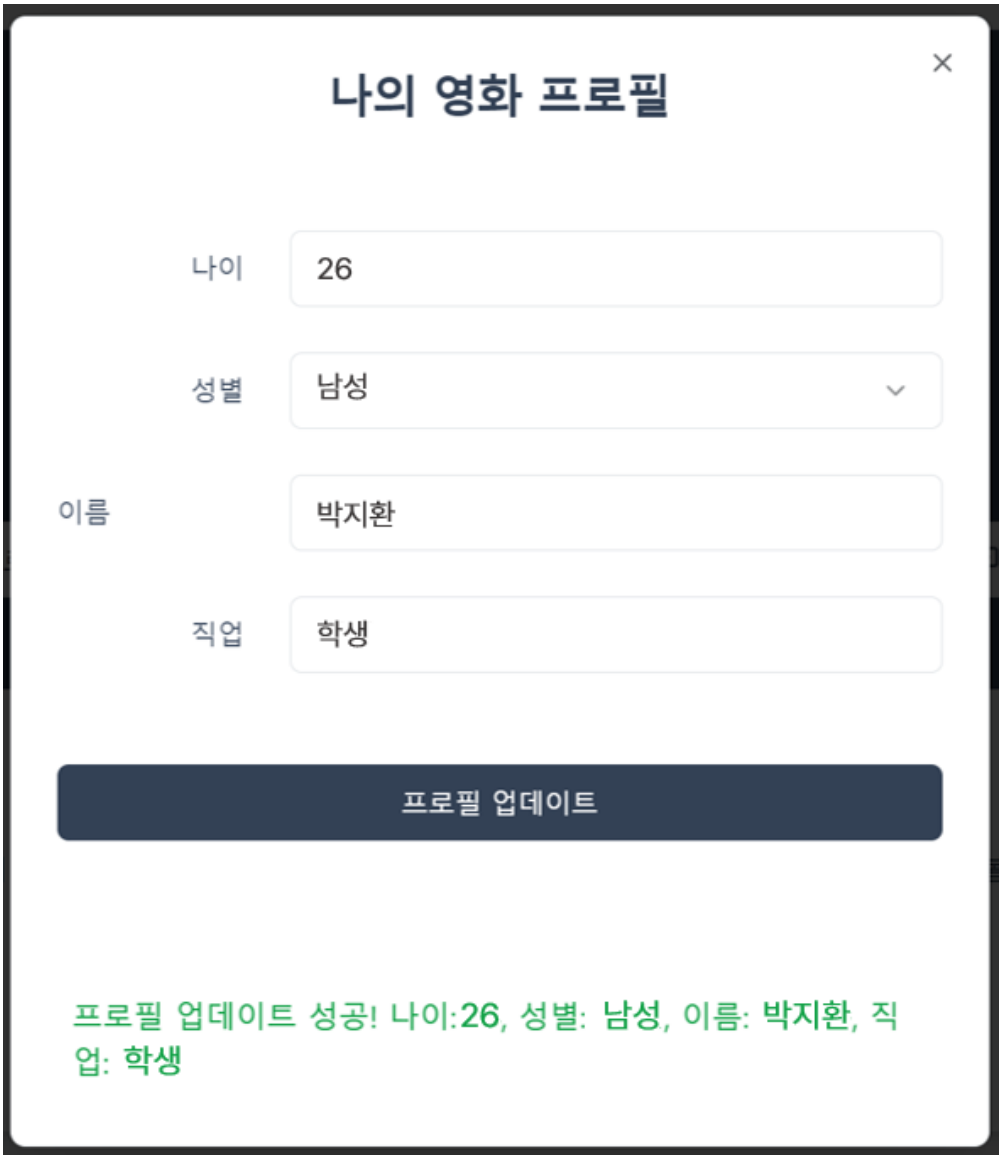

그림 10. 사용자 프로필 생성 화면

위와 같이 사용자의 나이, 성별과 이름, 직업을 별도 팝업에 입력하여 사용자 프로필을 생성하였다. 해당 프로필은 사용자 식별 ID값과 함께 데이터베이스의 users 테이블에 저장, 한 row를 구성하게 된다.

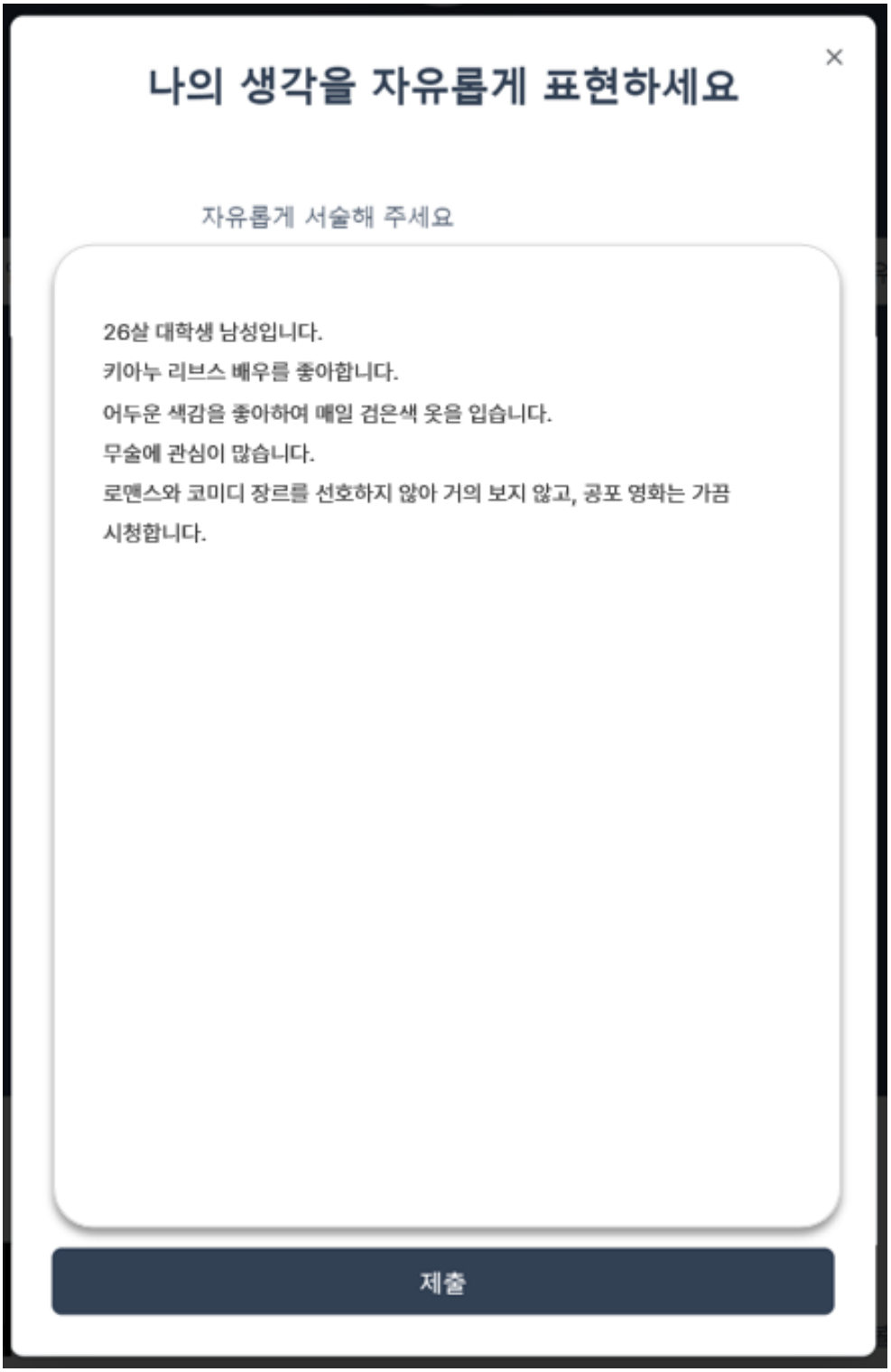

그림 11. 사용자 자유 입력 화면

그리고 자유 발화를 위와 같이 작성, 제출하여 프로필과 연동되는 사용자 페르소나를 생성하였다. 해당 테스트의 경우 '존 윅' 시리즈의 영화를 타겟으로 하여 선호 배우(키아누 리브스)와 비선호 장르(로맨스 및 코미디 비선호, 공포 중간 선호), 그리고 관심사(무술)와 색감

(검정색, 어두움) 등의 단서를 페르소나에 입력하였다. 언어 모델의 추론 능력 활용도를 테스트하기 위하여, 직접적인 선호 장르의 입력(액션) 대신 관심사(무술)에서 자연스럽게 선호할 만한 장르를 유추할 수 있도록 입력하였고, 실제로 영화에서 굉장히 어두운 색감의 장면이 많이 나오는 만큼, 데이터셋 내부 수치만으로는 표현할 수 없는 영화 내부의 분위기를 자력으로 추론하여 근거에 반영할 수 있는지 또한 테스트하기 위하여 자유 발화 내용을 위와 같이 구성하였다.

<추천 생성 및 추천 근거 제시>

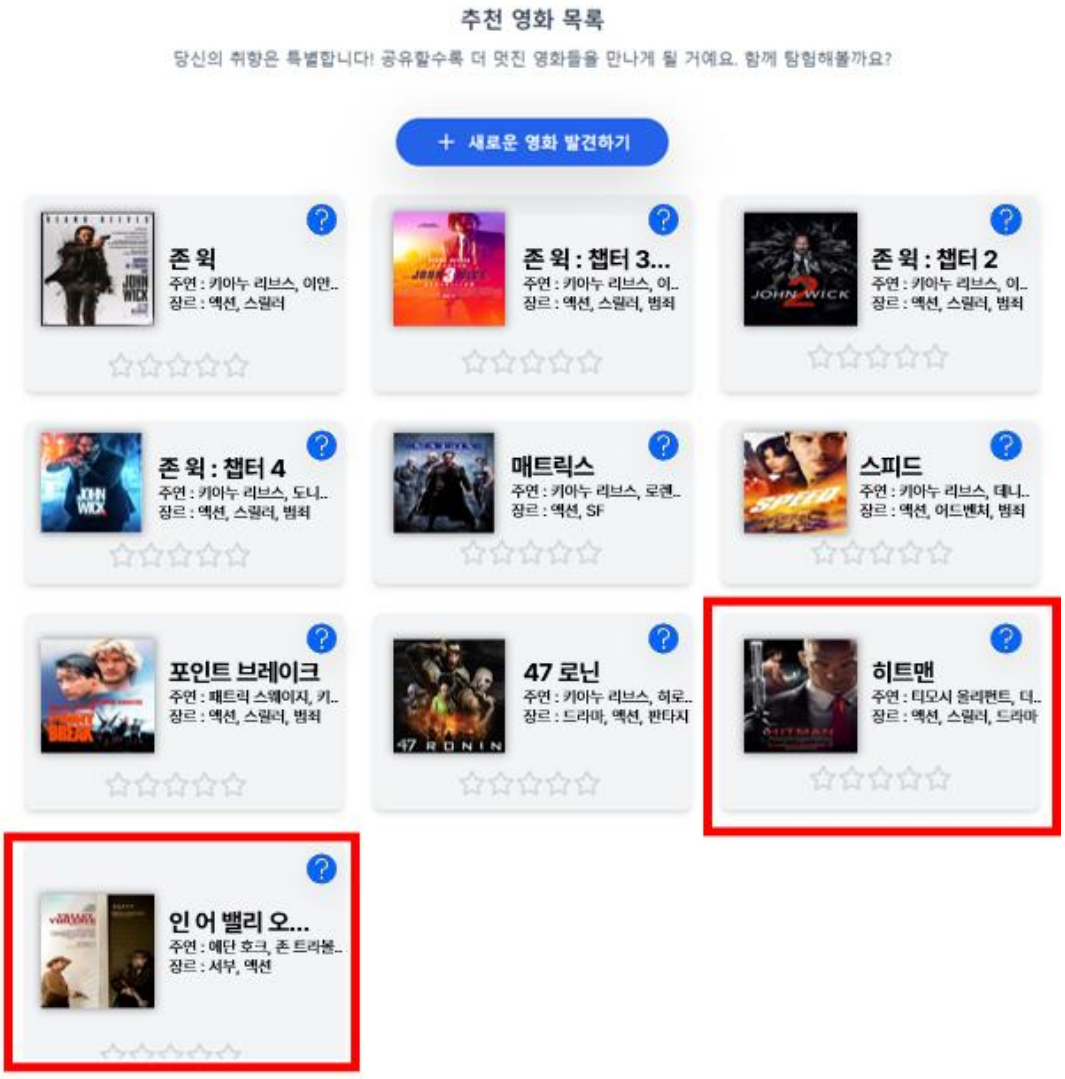

그림 12. 테스트 추천 결과 출력 화면

입력된 사용자 페르소나 정보를 기반으로 유사도를 계산, 언어 모델의 추론을 거쳐 총 10개의 영화가 추천되었다. 본래 답변으로 유도하고자 했던 존 윅 시리즈의 영화들이 상위권 순위에 위치해 있고, 그 아래로 선호 배우로 언급되었던 키아누 리브스 주연의 액션 영화들이

위치해 있다. 다만, 붉은 사각형으로 표시된 하위 2개 영화의 경우 장르를 제외하고는 발화 내용과의 연관성이 크지 않아 정확도 부분에서 개선의 여지가 있음을 확인할 수 있었다.

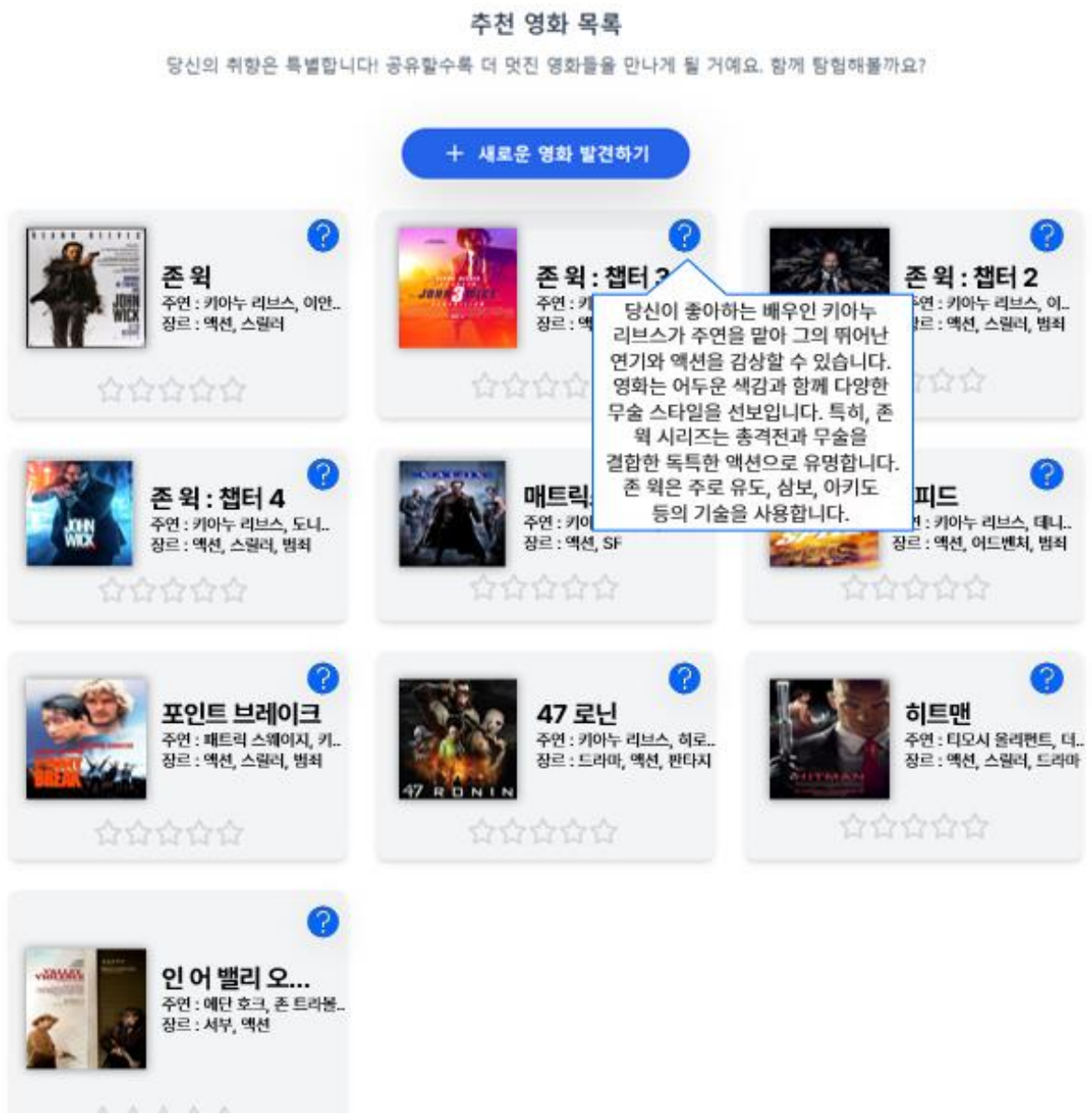

그림 13. 테스트 추천 근거 출력 화면

각 영화 블록 우측 상단의 물음표 이모티콘에 마우스 커서를 올릴 경우, 별도의 소형 탭이 hovering되어 추천에 대한 근거를 표시한다. 아래는 해당 추천에 사용된 사용자 발화 내용과 추천 근거의 비교이다.

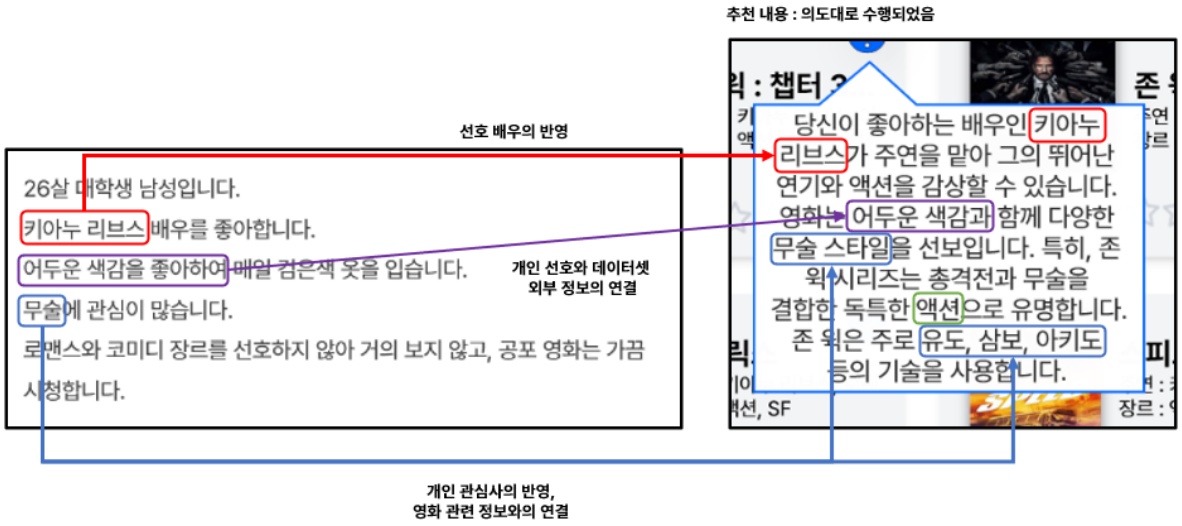

그림 14. 사용자 발화 내용과 추천 근거 요소의 비교

발화 내용에 직접적으로 표현된 선호 배우 정보의 반영이 잘 이루어졌고, 개인의 아이템 외적 선호(색감)과 데이터셋 외부 정보(영화 내 장면들의 전반적인 색감 배치)를 언어 모델의 정보를 기반으로 추론하여 연결한 근거 또한 제시가 되었다. 그리고 개인의 관심사(무술)를 기반으로 영화의 장르(액션)와 연결하여 제시한 근거 또한 일반적인 관점에서 받아들이기에는 무리가 없고, 또 자연스럽게 잘 표현되었다. 이러한 여러 가지 요소들을 고려하였을 때의 예시로 들 수 있는 페르소나 기반 추천 테스트의 결과 중 하나이다.

다만, 추가적으로 개선이 가능한 부분 또한 발견할 수 있었다. 우선, 하위 순위로 갈 수록 사용자의 발화 내용과 다소 연관성이 떨어지는 아이템이 선정이 되는 현상이 있어 추가적인 파라미터 조정이 필요한 부분으로 생각되었고, 언어 모델의 경우 데이터셋 내부에서 찾을 수 없는 정보가 추론에 필요한 경우 그 정확성을 높이는 부분이 매우 중요하다고 생각되는데, 이러한 맥락에서 현재 활발히 연구가 진행되고 있는 RAG(Retrieval-Augment Generation, 검색 증강 생성) 기법을 적용한 LLM을 본 시스템에 도입한다면 해당 경우에 대한 정확성을 큰 폭으로 높일 수 있을 것으로 예상된다.

5. 결론 및 향후 연구 방향

하이브리드 추천 시스템과 LLM을 결합하여 사용자의 발화를 분석하여 이를 근거로 한 추천을 제공하는 프레임워크를 구축할 수 있었다. 연구를 진행하는 과정에서 CRS에서의 LLM 적용 분야 및 프롬프트 설계에 대한 아이디어를 얻을 수 있었고, 이를 기반으로 하여 multi-turn 대화에서 전체 맥락을 반영하는 프롬프트 설계 및 RAG(Retrieval-Augment Generation, 검색 증강 생성) 기반 LLM의 적용 등의 개선 방안을 통하여 사용자의 개인 선호에 더 일치하는, 검증된 정보만을 제공하는 최적의 영화 추천 플랫폼을 개발할 것이다.

연산 자원 등의 현실적 제약 조건으로 인하여 상기한 개선 방안을 본 연구에서 적용하지 못한 부분이 아쉬움으로 남지만, 추가적인 연구를 통하여 계속해서 플랫폼을 개량해 나갈 것이고, 그 과정에서 얻은 지식과 최적의 프레임워크를 주제로 하는 논문을 작성해 볼 계획이다.

6. 참고 문헌

[1] Y. Feng, S. Liu, Z. Xue et al. "A Large Language Model Enhanced Conversational Recommender System" Aug, 2023.
[2] L. Friedman, S. Ahuja, D. Allen et al. "Leveraging Large Language Models in Conversational Recommender Systems", May. 2023
[3] X. Wang, X. Tang, W.X. Zhao et al. "Rethinking the Evaluation for Conversational Recommendation in the Era of Large Language Models". EMNLP 2023. Nov. 2023.