Next-Gen Recommender System: LLMs for Movie Discovery and Genre Classification

Report submitted to the SASTRA Deemed to be University as the requirement for the course

INT317 - Data Mining and Analytics

Submitted by

Sanjai S - 126018042

October 2025

SCHOOL OF COMPUTING

THANJAVUR, TAMIL NADU, INDIA – 613 401

List of Figures

Abstract

Recommender systems are really important for helping users find their way through the h selectionof contentavailableon streamingplatforms.However,thesesystemshave some challenges. One major issue is the cold start problem, where new users and items don't have any interaction data yet. There's also data sparsity, which makes it hard to recommend less popular content. Traditional methods like Collaborative Filtering (CF) and Content-Based Filtering (CBF) often don't take advantage of valuable contextual information, like movie subtitles, and they can struggle with scalability and overfitting in deep learning models. Because of these challeng there's a need for approaches that are more flexible and context-aware. This project looks at how Large Language Models (LLMs) like GPT-4 can help solve these problems. The idea is to extract subtitles from movie trailers and use them to create summaries. These summaries are then used predicting genres and making personalized recommendations. Their ability to learn from very litt data, LLMs can understand user preferences even when there's not much previous interaction da Results from experiments show that LLMs perform better than traditional methods in genr prediction, using metrics like precision, recall, and F1 scores, as well as in recommendatio measured by metrics like MAP, MRR, and NDCG. By combining subtitles and summaries, this method enhances personalization and can effectively scale for real-world use. Looking ahead, the focus will be on improving these models for larger datasets and making them suitable for real-tir applications.

Keywords: RecommenderSystems, Large Language Models (LLMs), GPT-4, Movie Summarization, Genre Classification

Table of Contents

CHAPTER 1:

SUMMARY OF THE BASE PAPER

Title	: Transformative Movie Discovery: Large Language Models for Recommendation and Genre Prediction
Authors	: Subham Raj, Anurag Sharma, Sriparna Saha, Brijraj Singh, Niranjan Pedanekar
Journal name	: IEEE Access
Publisher	: IEEE
Year	: 2024
Indexed in	: Scopus

Content:

Personalized movie suggestions and genre prediction have become essential for raising us pleasure and engagement in the age of digital streaming platforms. The necessity for eff content recommendation systems has grown in importance due to the proliferation of OTT (Ove The-Top) services such as Netflix, Amazon Prime Video, and Disney+. This study offers a fresh method for finding films that makes use of massive language models. We investigate the transformative potential of these algorithms in two main areas: recommendation and gen prediction. We used several prompting strategies to assess four distinct LLMs across the MovieLens-100K dataset. To provide large language models (LLMs) with cues, we used audio subtitles from movie trailers. From these questions, we created movie summaries, which we the utilized as extra prompts to identify different genres of films. Additionally, the history of each user's interactions with films is supplied for the recommendation problem, together with generated summary information that goes with it. Large language models are excellent at captu intricate user preferences and film attributes, according to our tests, producing incredibly precis and tailored movie suggestions. Ranking measures show that our method routinely performs be in movie recommendation tasks than the current supervised baseline methods

ARCHITECTURE:

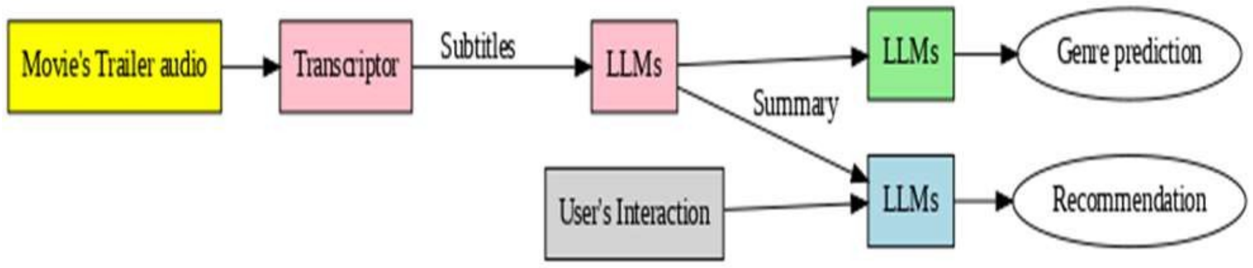


Fig.1.1 – Proposed Architecture

The main steps are:

1. Data Extraction & Preprocessing: Extract and preprocess movie transcripts using YouTube-transcript-API and YouTube-Whisperer.

2. Summary Generation: Generate context-rich movie summaries using Zero-Shot and Few Shot prompting techniques.

3. Genre Prediction: Predict movie genres by constructing prompts based on the generat summaries.

4. Recommendation:Provide personalized movie recommendations by integrating user interaction history with summary and genre metadata.

5. Evaluation: Evaluate summaries using ROUGE, genres using Precision/Recall/F1-score, and recommendations using MAP, MRR, and NDCG scores.

CHAPTER 2

MERITS AND DEMERITS OF THE BASE PAPER

MERITS:

- Innovative Use of LLMs for Recommendation Systems: The paper leverages the advanced capabilities of LLMs, such as GPT-3.5 Turbo, GPT-4, Mistral 7B, and Deepseek 7B, to address challenges in movie recommendation and genre prediction. By utilizing audio subtitles from mov trailers as prompts, the approach captures nuanced contextual information, surpassing tradition methods like collaborative filtering and deep neural networks (DNNs). This innovative application demonstrates LLMs' potential to model complex user preferences and movie characteristics effectively.

- Effective Handling of Cold-Start Problem: The methodology mitigates the coldstart problem, a significant limitation in traditional recommender systems, by generating movie summaries predictinggenres from trailer subtitles. This allows the system to provide meaningful recommendations for new movies or users with limited interaction data, enhancing its applicabili in real-world scenarios where data sparsity is common.

- Robust Evaluation Metrics: The study employs a comprehensive set of evaluation metrics, including ROUGE for summary generation, precision, recall, and F1-score for genre prediction, ar ranking metrics like Mean Average Precision (MAP), Mean Reciprocal Rank (MRR), and Normalized Discounted Cumulative Gain (NDCG) for recommendations. Additionally, human evaluation metrics (relevance, fluency, consistency, and storyline recall) ensure a thorough assessment of summary quality, adding credibility to the results.

- Superior Performance Over Baselines: Experimental results on the MovieLens100K dataset show that LLMs, particularly GPT-4, outperform state-of-the-art deep learning models like GCA and multitasking-based recommender systems in ranking prediction tasks. This highlights t superiority of LLMs in capturing semantic relationships and generalizing across sparse data, setti a new benchmark for recommendation systems.

- Versatility with Zero-Shot and Few-Shot Learning: The use of zero-shot and fewshot prompting enables the system to adapt to new tasks without extensive retraining. This versatilit makes the approach scalable and applicable to various domains beyond movies, such as commerce or content streaming, where minimal task-specific data is available.

DEMERITS:

- Limited Dataset Scope: The study uses only MovieLens-100K, a small dataset. This restricts scalability and generalizability evaluation.

- Token Constraints: Token limits restrict recommendation tasks to 20 movies, limiting preferenc modeling. This affects real-world applicability.

- Subtitle Quality Dependence: The method relies on high-quality subtitle extraction. Errors could reduce summary and genre prediction accuracy.

- Limited LLM Contribution: Deepseek 7B and Mistral 7B are restricted to partial tasks. This limits comparative analysis of their capabilities.

- Lack of Cost Analysis: The paper omits detailed computational cost insights. This is critical for scalability in resource-constrained settings.

CHAPTER 3

SOURCE CODE

1. SUBTITLE EXTRACTION

```
import os
import csv
import whisper
import yt_dlp
import multiprocessing
from youtube_transcript_api import YouTubeTranscriptApi, TranscriptsDisabled,
NoTranscriptFound
import torch.multiprocessing as mp
from youtubesearchpython import VideosSearch
import time
import pandas as pd

df = pd.read_csv("u_item.csv", encoding="ISO-8859-1")
movie_titles = df["movie title"].tolist()

CSV_FILE = "movies_sub.csv"


def search_trailer(movie):
    """Searches for the official trailer on YouTube and returns the video ID."""
    try:
        search = VideosSearch(f"{movie} official trailer", limit=1)
        result = search.result()
        if result["result"]:
            return result["result"][0]["id"]
    except Exception as e:
        print(f"Error searching trailer for {movie}: {e}")
    return None


def get_transcript_youtube(video_id):
    """Fetches subtitles from YouTube API."""
    try:
        transcript = YouTubeTranscriptApi.get_transcript(video_id)
        return " ".join([t["text"] for t in transcript])
    except (TranscriptsDisabled, NoTranscriptFound):
        return None
    except Exception as e:
        print(f"Error fetching subtitles for {video_id}: {e}")
        return None
```

```
def download_audio(video_id, output_file):
    """Downloads audio from YouTube for Whisper transcription."""
    try:
        video_url = f"https://www.youtube.com/watch?v={video_id}"
        ydl_opts = {
            "format": "bestaudio/best",
            "outtmpl": output_file,
            "postprocessors":  [{"key":  "FFmpegExtractAudio",  "preferredcodec":  "mp3",
"preferredquality": "192"}],
            "quiet": True,
            "ffmpeg_location": r"C:\Program Files\ffmpeg\bin",
        }
        with yt_dlp.YoutubeDL(ydl_opts) as ydl:
            ydl.download([video_url])
        print(f"Audio downloaded for {video_id}")
    except Exception as e:
        print(f"Error downloading audio for {video_id}: {e}")


def read_existing_movies():
    """Reads the existing CSV file and returns a set of completed movies."""
    if not os.path.exists(CSV_FILE):
        return set()
    try:
        with open(CSV_FILE, "r", encoding="utf-8") as f:
            return {row[1] for row in csv.reader(f) if row}
    except Exception as e:
        print(f"Error reading CSV file: {e}")
        return set()


def write_to_csv(index, movie, transcript):
    """Writes transcript to CSV."""
    try:
        with open(CSV_FILE, "a", newline="", encoding="utf-8") as f:
            writer = csv.writer(f)
            writer.writerow([index, movie, transcript])
    except Exception as e:
        print(f"Error writing to CSV: {e}")


def process_movie(index, movie):
    """Processes a movie to extract its transcript."""
    print(f"Processing: {movie}")
    video_id = search_trailer(movie)
    if not video_id:
        write_to_csv(index, movie, "Trailer not found")
        return
    transcript = get_transcript_youtube(video_id)
    if transcript:
        write_to_csv(index, movie, transcript)
        return

    print(f"No subtitles found using YouTube API. Downloading audio for : {movie}")
    if not os.path.exists(f"{index}.mp3.mp3"):
        audio_file = f"{index}.mp3"
        download_audio(video_id, audio_file)
        write_to_csv(index, movie, f"Audio downloaded: {audio_file}")  # Whisper handled
separately


def get_trailer_transcripts_parallel(movies, num_workers=4):
    """Parallelized processing of YouTube subtitle extraction and audio downloads."""
    completed_movies = read_existing_movies()
    movies_to_process = [(i, movie) for i, movie in enumerate(movies, start=1) if movie not in
completed_movies]

    with multiprocessing.Pool(num_workers) as pool:
        pool.starmap(process_movie, movies_to_process)


if __name__ == "__main__":
    mp.set_start_method("spawn", force=True)
    movies = movie_titles
    get_trailer_transcripts_parallel(movies, num_workers=4)
    print(f"Transcripts saved to {CSV_FILE}")
```

2. PREPROCESSING

```
## First Level Pre-Processing

u_item = pd.read_csv("u_item.csv", encoding="ISO-8859-1")

u_item.info()

duplicate_movie_titles = u_item[u_item.duplicated(subset=['movie title'], keep=False)]
print(duplicate_movie_titles[['movie id', 'movie title']])

movie_dict = {}
for index, row in u_item.iterrows():
    movie_title = row['movie title']
    movie_id = row['movie id']
    if movie_title not in movie_dict:
        movie_dict[movie_title] = movie_id
movies = pd.read_csv('movie_sub.csv')
```

```
movies.columns

movies = movies.drop_duplicates(subset='Movie_Name', keep='first')
movies.insert(0,'Movie_ID', movies['Movie_Name'].map(movie_dict))
movies

movies = movies.dropna(subset=['Movie_ID'])
movies['Movie_ID'] = movies['Movie_ID'].astype(int)

to_process = data[data['Subtitle Transcript'].str.contains("Audio downloaded", na=False)]
to_process

!pip install --upgrade --no-cache-dir openai-whisper

import whisper
import os

transcript = {}

model = whisper.load_model("medium").to("cuda")

def write_to_csv(index, movie, transcript):
    for index, new_transcript in transcript.items():
            to_process.loc[to_process.Movie_ID == index, "Subtitle Transcript"] = new_transcript
    print(f"SUBTITLE UPDATED FOR: {index} :: {movie}")

def transcribe_audio(file_path):
    result = model.transcribe(file_path)
    return result["text"]

for y, z in to_process.iterrows():
    print(f"\n Processing: {z.loc['Movie_ID']}::{z.loc['Movie_Name']}")  # Use .loc
    audio_file = f"{z.loc['Movie_ID']}.mp3.mp3"  # Use .loc
    try:
            transcript[z.loc['Movie_ID']] = transcribe_audio(audio_file)  # Use .loc
    except Exception as e:
            print(f"{z.loc['Movie_ID']} TRANSCRIPTION FAILED: {e}")  # Print exception
            continue

print("All transcripts updated successfully!")

for y,z in to_process.iterrows():
    write_to_csv(z[0],z[1],transcript)

## Second Level Pre-Processing

data = pd.read_csv('movies_sub.csv')
data
```

```
data.shape

data.drop(columns=[col for col in data.columns if col.startswith('Unnamed:')], inplace=True)

import re

def non_english(text):
    if not isinstance(text, str) or text.isspace():
        return False
    return bool(re.search(r'[^\x00-\x7F]+', text))

non_english_movies = data[data['Subtitle Transcript'].apply(non_english)]

print("Movies with potentially non-English subtitles:")
non_english_movies


!pip install deep-translator

from deep_translator import GoogleTranslator

translator = GoogleTranslator(source='auto', target='en')

def translate_text(text):
    try:
        if isinstance(text, str) and text.strip():
            return translator.translate(text)
        return text
    except:
        return text

non_english_movies["Subtitle Transcript"] = non_english_movies["Subtitle Transcript"].astype(str).apply(translate_text)

non_english_movies

data.update(non_english_movies)
data

## Third Level Pre-Processing

exceeding = data[data['Subtitle Transcript'].str.split().str.len() > 600]
exceeding

def shorten_transcript(transcript):
    words = transcript.split()
    if len(words) > 600:
        return " ".join(words[:600])
    return transcript
```

```
exceeding['Subtitle Transcript'] = exceeding['Subtitle Transcript'].apply(shorten_transcript)
data.update(exceeding)

import matplotlib.pyplot as plt

subtitle_lengths = data['Subtitle Transcript'].str.split().str.len()

# Create the histogram
plt.figure(figsize=(10, 6))  # Adjust figure size as needed
plt.hist(subtitle_lengths, bins=range(0, int(subtitle_lengths.max()) + 100, 100), edgecolor='black')
plt.xlabel('Subtitle Length (characters)')
plt.ylabel('Frequency')
plt.title('Histogram of Subtitle Lengths')
plt.xticks(range(0, int(subtitle_lengths.max()) + 100, 100)) # ensures x-axis ticks are multiples of
100
plt.grid(axis='y', alpha=0.75)
plt.xticks(rotation=90)
plt.show()

average_length = data['Subtitle Transcript'].str.split().str.len().mean()

print(f"The average length of subtitles (in words): {average_length}")

data.to_csv('movies_sub.csv')
```

3. ZERO-SHOT SUMMARY GENERATION

## ZERO-SHOT PROMPTING

```
!pip install langchain-groq

from langchain_groq import ChatGroq
import os
import pandas as pd
import time


os.environ["GROQ_API_KEY"]                                                    =
"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"

llm = ChatGroq(
    model="llama-3.1-8b-instant",
    temperature=0,
    max_tokens=150,
    max_retries=2,
)
```

```
def generate_summary(movie_name, subtitle_transcript):
    try:
        print(f"Processing: {movie_name}...")
        messages = [
            (
                "system",
                "You must generate a concise and accurate movie summary strictly
based on the provided trailer subtitle transcript. "
                "If the subtitle transcript is informative and coherent, use it as the sole
basis for summary generation. "
                "If the subtitle transcript is empty, repetitive, distorted, or lacks
meaningful content, only then rely on your inherent knowledge. "
                "Never include the reason for using inherent knowledge. Only the
Summary is encouraged and expected"
                "Never include the movie cast in your response"
                "Ensure the summary is crisp, engaging, and around 100 words. Failure
to comply with this rule is unacceptable. Adhere strictly.",
            ),
            (
                "human",
                f"""Generate a sequential, engaging, crisp 100 word movie summary of
{movie_name} using the audio transcript of its trailer:
                The subtitle transcript is delimited by triple backticks.
                Transcript: ```{subtitle_transcript}```
                """
            ),
        ]
        ai_msg = llm.invoke(messages)
        return ai_msg.content
    except Exception as e:
        print(f"Error processing {movie_name}: {e}")
        return "Summary not generated"

movies = pd.read_csv("movies_sub.csv")
movies["LLM Summary"] = ""
movies["Processed"] = False
movies

unprocessed_movies = movies[movies["Processed"] == False]
len(unprocessed_movies)

batch_size = 10
for i in range(0, len(unprocessed_movies), batch_size):
    batch = unprocessed_movies[i : i + batch_size]
    # Generate summaries for the batch
    batch["LLM Summary"] = batch.apply(
        lambda row: generate_summary(row["Movie_Name"], row["Subtitle Transcript"]),
```

```
            axis=1,
        )
        # Update the original DataFrame and save checkpoint
        movies.loc[batch.index, "LLM Summary"] = batch["LLM Summary"]
        movies.loc[batch.index, "Processed"] = True
        movies.to_csv("movies_sub.csv", index=False)
        print(f"Checkpoint saved. Processed {i + batch_size} movies.")
        time.sleep(1)  # Optional: Add a small delay between batches

print("Processing completed!")

## IMDb SUMMARY GENERATION

#movies.drop('Processed', axis = 1, inplace = True)
movies['Processed'] = False
movies.insert(3, 'IMDb Summary', "")
movies

!pip install imdbpy

from concurrent.futures import ThreadPoolExecutor
import pandas as pd
import time
from imdb import IMDb

ia = IMDb()

def fetch_movie_data(index, title):
        try:
            print(f"Processing: {index} --> {title}")

            if movies.loc[movies['Movie_ID'] == index, 'Completed'].values[0]:
                return

            search_results = ia.search_movie(title)

            if search_results:
                movie_id = search_results[0].movieID
                movie = ia.get_movie(movie_id)
                synopsis = movie.get('plot outline')

                movies.loc[movies['Movie_ID'] == index, ['IMDb Summary', 'Completed']] =
[synopsis, True]

                if index % 50 == 0:
                    movies.to_csv('movies_sub.csv', index=False)
                    print(f"Progress saved at {index}")

            else:
                print(f"Movie '{title}' not found.")

            time.sleep(0.5)

    except:
            movies.loc[movies['Movie_ID'] == index, ['IMDb Summary', 'Completed']] = ['N/A',
False]


with ThreadPoolExecutor(max_workers=5) as executor:
    for idx, row in movies.iterrows():
            executor.submit(fetch_movie_data, row['Movie_ID'], row['Movie_Name'])

# Final save
movies.to_csv('movies_sub.csv', index=False)
print("Final data saved to 'movies_sub.csv'")


movies.to_csv('movies_sub.csv', index = False)
```

4. FEW-SHOT SUMMARY GENERATION

```
## FEW-SHOT PROMPTING

import pandas as pd

movies = pd.read_csv("movies_sub.csv")
movies["Llama Summary Few Shot"] = ""
movies["Generated"] = False
movies.drop('Unnamed: 0', axis = 1, inplace = True)

from langchain_groq import ChatGroq
import os
import pandas as pd
import time


os.environ["GROQ_API_KEY"]                                                  =
"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"

llm = ChatGroq(
    model="llama-3.1-8b-instant",
    temperature=0,
    max_tokens=150,
    max_retries=2,
)

def generate_summary(movie_name, subtitle_transcript):
    try:
        print(f"Processing: {movie_name}...")  # Print current movie

        # IMDb Few-Shot Examples (Fixed for consistency)
        imdb_examples = """
        Movie_Name: Inception (2010)
        IMDb Synopsis: A thief who steals corporate secrets through the use of dream-sharing
technology is given the inverse task of planting an idea into the mind of a C.E.O., but his tragic pa
may doom the project and his team to disaster.

        Movie_Name: Interstellar (2014)
        IMDb Synopsis: When  Earth becomes uninhabitable in the future, a farmer and  ex-
NASA pilot, Joseph Cooper, is tasked to pilot a spacecraft, along with a team of researchers, to fin
a new planet for humans.

        Movie_Name: The Shawshank Redemption (1994)
        IMDb Synopsis: A banker convicted of uxoricide forms a friendship over a quarte
century with a hardened convict, while maintaining his innocence  and trying to remain hopeful
through simple compassion.
        """

        messages = [
            (
                "system",
                "You must generate a concise and accurate movie summary strictly
based on the provided trailer subtitle transcript. "
                "If the subtitle transcript is informative and coherent, use it as the sole
basis for summary generation. "
                "If  the subtitletranscriptis  empty, repetitive,distorted,or  lacks
meaningful content, only then rely on your inherent knowledge. "
                "Never include the reason for using inherent knowledge. Only the
Summary is encouraged and expected. "
                "Never include the movie cast in your response. "
                "Ensure the summary is crisp, engaging, and around 100 words. Failure
to comply with this rule is unacceptable. Adhere strictly.",
            ),
            (
                "human",
                f"""Here are examples of movie summaries from IMDb:

                {imdb_examples}

                Now, generatea  sequential,engaging,and crisp 100-wordmovie
summary of {movie_name} using the audio transcript of its trailer:
                The subtitle transcript is delimited by triple backticks.
                Transcript: ```{subtitle_transcript}```
                """
            ),
        ]
```

```
                ai_msg = llm.invoke(messages)
                return ai_msg.content

        except Exception as e:
                print(f"Error processing {movie_name}: {e}")
                return "Summary not generated"


unprocessed_movies = movies.loc[(movies['Generated']==False)]

batch_size = 10
for i in range(0, len(unprocessed_movies), batch_size):
        batch = unprocessed_movies[i : i + batch_size]
        batch["Llama Summary Few Shot"] = batch.apply(
                lambda row: generate_summary(row["Movie_Name"], row["Subtitle Transcript"]),
                axis=1,
        )

        movies.loc[batch.index, "Llama Summary Few Shot"] = batch["Llama Summary Few Shot"]
        movies.loc[batch.index, "Generated"] = True
        movies.to_csv("movies_sub.csv", index=False)
        print(f"Checkpoint saved. Processed {i + batch_size} movies.")
        time.sleep(1)

print("Processing completed!")

movies.to_csv("movies_sub.csv")
```

5. GENRE PREDICTION

```
## GENRE PREDICTION USING LLM

!pip install langchain-groq

from langchain_groq import ChatGroq
import os
import pandas as pd
import time

os.environ["GROQ_API_KEY"]                                        =
"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"

llm = ChatGroq(
        model="llama-3.1-8b-instant",
        temperature=0,
        max_tokens=100,
        max_retries=2,
)

def predict_genres(movie_name, movie_summary):
    try:
        print(f"Processing: {movie_name}...")

        genres_list = [
            "Action",  "Adventure", "Animation", "Children's", "Comedy", "Crime",
"Documentary", "Drama", "Fantasy",
            "Film Noir", "Horror", "Musical", "Mystery", "Romance", "Sci-Fi", "Thriller",
"War", "Western"
        ]

        messages = [
            (       "system",
                    "You are an expert in movie genre classification. Your task is to classify
the movie summary into relevant genres based on the predefined list of genres. "
                    "The predefined genres are: " + ", ".join(genres_list) + ". "
                    "Only return the list of genres in Python list format, without any
additional text."),
            (
                    "human",
                    f"""Given the following movie name and summary, return the genres as
a Python list:
                    Movie Name: {movie_name}
                    Summary: {movie_summary}
                    Genres: """
            )
        ]

        ai_msg = llm.invoke(messages)
        return ai_msg.content.strip()

    except Exception as e:
        print(f"Error predicting genres: {e}")
        return "[]"  # Return an empty list if an error occurs

movies = pd.read_csv("movies_sub.csv")

genres_list =   ["Action",  "Adventure", "Animation", "Children's", "Comedy", "Crime",
"Documentary", "Drama", "Fantasy","Film Noir", "Horror", "Musical", "Mystery", "Romance",
"Sci-Fi", "Thriller", "War", "Western"]

unprocessed_movies = movies[movies["Processed"] == False]
len(unprocessed_movies)

batch_size = 10
for i in range(0, len(unprocessed_movies), batch_size):
    batch = unprocessed_movies[i : i + batch_size]
    # Predict genres once per movie
    batch["Predicted  Genres"]  =  batch.apply(lambda  row:  predict_genres(row["Movie_Name"],
row["Llama Summary Few Shot"]), axis=1)

    # Convert predicted genres to one-hot encoding
    for genre in genres_list:
            batch[genre] = batch["Predicted Genres"].apply(lambda genres: 1 if genre in genres els
0)

    # Update the original DataFrame and save checkpoint
    movies.loc[batch.index, genres_list] = batch[genres_list]
    movies.loc[batch.index, "Processed"] = batch[genres_list].sum(axis=1) > 0  # Mark as True if
any genre is assigned
    movies.to_csv("movies_sub.csv", index=False)

    print(f"Checkpoint saved. Processed {i + batch_size} movies.")
    time.sleep(1)  # Optional: Add a small delay between batches

print("Processing completed!")
```

6. RECOMMENDATION TASK

```
!pip install langchain_groq

!pip install rank-metrics

import pandas as pd
import numpy as np


from langchain_groq import ChatGroq
import os
import pandas as pd

# Set your API key
os.environ["GROQ_API_KEY"]                                                          =
"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"

# Init LLM
llm = ChatGroq(
    model="llama-3.1-8b-instant",
    temperature=0,
    max_tokens=100,
    max_retries=2,
)
```

```
# Load datasets
df = pd.read_csv('u.csv')
movies_sub = pd.read_csv('movies_sub.csv')

# Load previously saved predictions (if any)
csv_file = "predicted_rankings.csv"
predicted_rankings = {}

if os.path.exists(csv_file):
    existing_df = pd.read_csv(csv_file)
    predicted_rankings                    =                    dict(zip(existing_df["user_id"],
existing_df["predicted_ranking"].apply(eval)))

# Buffer for batch saving
buffer = []

# Iterate over users
for idx, user_id in enumerate(df['user id'].unique()):
    if user_id in predicted_rankings:
        continue

    try:
        user_movies = df[df['user id'] == user_id]
        if len(user_movies) < 20:
            continue

        sorted_movies = user_movies.sort_values(by='rating', ascending=False)

        top_15 = sorted_movies.head(15)

        remaining_5 = sorted_movies.tail(5)

        top_15_summaries    =    movies_sub[movies_sub['Movie_ID'].isin(top_15['item
id'])][['Movie_ID', 'Llama Summary Few Shot']]

        remaining_5_summaries                                                    =
movies_sub[movies_sub['Movie_ID'].isin(remaining_5['item id'])][['Movie_ID', 'Llama Summary
Few Shot']]

        if len(top_15_summaries) < 15 or len(remaining_5_summaries) < 5:
            continue

        prompt = "The following are 15 movies the user has liked the most, ranked from most
liked to least liked. Learn the user's taste in terms of genres and themes based on their summari

        for _, row in top_15_summaries.iterrows():
            prompt  +=  f"Movie:  {row['Movie_ID']},  Summary:  {row['Llama  Summary
Few Shot']}\n"
```

```
        prompt += "\nNow, based on the user's genre and theme preferences learned above, r
the following 5 movies from most liked to least liked:\n"

        for _, row in remaining_5_summaries.iterrows():
            prompt  +=  f"Movie:  {row['Movie_ID']},  Summary:  {row['Llama  Summary
Few Shot']}\n"

        prompt += (
            "\nReturn ONLY a Python list of Movie IDs in ranked order like this:\n"
            "[173, 257, 110, 13, 580]\n"
            "DO NOT include any other text or explanation."
        )

        # Get response
        ai_msg = llm.invoke([
            ("system", "You are a movie recommendation engine. Rank only. Return only a
list."),
            ("human", prompt)
        ])

        output = ai_msg.content.strip()
        if output.startswith("[") and output.endswith("]"):
            predicted = [int(x.strip()) for x in output[1:-1].split(",")]

            true_sorted =    remaining_5[['itemid',    'rating']].sort_values(by='rating',
ascending=False)
            ground_truth = true_sorted['item id'].tolist()

            if sorted(predicted) != sorted(ground_truth):
                print(f"Movie mismatch for user {user_id}, skipping...")
                continue

            predicted_rankings[user_id] = predicted
            buffer.append([user_id, predicted])
            print(f"User {user_id} predicted")

            # Write every 10 users
            if len(buffer) >= 10:
                df_to_save     =       pd.DataFrame(buffer,   columns=["user_id",
"predicted_ranking"])
                if not os.path.exists(csv_file):
                    df_to_save.to_csv(csv_file, index=False)
                else:
                    df_to_save.to_csv(csv_file,      mode='a',       header=False,
index=False)
                buffer.clear()

                # Print changes saved
                total_predicted = len(predicted_rankings)
```

```
                        print(f"Changes saved! Total movies predicted: {total_predicted}")

                else:
                    print(f" Invalid format for user {user_id}: {output}")

        except Exception as e:
            print(f"Error for user {user_id}: {e}")

# Save remaining predictions at the end
if buffer:
    df_to_save = pd.DataFrame(buffer, columns=["user_id", "predicted_ranking"])

    if not os.path.exists(csv_file):
        df_to_save.to_csv(csv_file, index=False)
    else:
        df_to_save.to_csv(csv_file, mode='a', header=False, index=False)

    # Print final changes saved
    total_predicted = len(predicted_rankings)
    print(f"Changes saved! Total movies predicted: {total_predicted}")

print("\nAll users finished!")


len(df['user id'].unique())

len(predicted_rankings)

# Load your data
df = pd.read_csv("u.csv")
predicted_df = pd.read_csv("predicted_rankings.csv")

# Create dictionary for ground truths
ground_truth_rankings = {}

for user_id in predicted_df["user_id"]:
    user_movies = df[df['user id'] == user_id]
    if len(user_movies) < 20:
        continue

    sorted_movies = user_movies.sort_values(by='rating', ascending=False)
    remaining_5 = sorted_movies.tail(5)

    true_sorted = remaining_5[['item id', 'rating']].sort_values(by='rating', ascending=False)
    ground_truth = true_sorted['item id'].tolist()
    ground_truth_rankings[user_id] = ground_truth

print(f"Ground truth reconstructed for {len(ground_truth_rankings)} users")
```

```
ground_truth_rankings
```

7. Distil-BERT

```
## MODEL DOWNLOAD

from transformers import DistilBertTokenizer, DistilBertModel

model_name = "distilbert-base-uncased"

# Downloads & caches locally (usually in ~/.cache/huggingface/)
tokenizer = DistilBertTokenizer.from_pretrained(model_name)
model = DistilBertModel.from_pretrained(model_name)

tokenizer.save_pretrained("./distilbert_local")
model.save_pretrained("./distilbert_local")

## DATASET LOADING

import pandas as pd
import numpy as np
import torch
from transformers import DistilBertTokenizerFast, DistilBertForSequenceClassification, Trainer
TrainingArguments
from transformers import DistilBertTokenizer, DistilBertModel
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.preprocessing import MultiLabelBinarizer

movies = pd.read_csv('movies_sub.csv')
u_item = pd.read_csv("u_item.csv", encoding="ISO-8859-1")

u_item.columns

u_item.rename(columns={'movie id': 'Movie_ID'}, inplace=True)

genre_cols = ['Action', 'Adventure', 'Animation', "Children's", 'Comedy',
              'Crime', 'Documentary', 'Drama', 'Fantasy', 'Film Noir',
              'Horror', 'Musical', 'Mystery', 'Romance', 'Sci-Fi',
              'Thriller', 'War', 'Western']

data = movies[['Movie_ID', 'Movie_Name', 'Llama Summary Few
Shot']].merge(u_item[['Movie_ID'] + genre_cols], on='Movie_ID')
data.head()

#data = data.sample(n=100, random_state=42).reset_index(drop=True)
```

```
texts = data['Llama Summary Few Shot'].tolist()
labels = data[genre_cols].values


## MODEL IMPLEMENTATION

device = "cuda" if torch.cuda.is_available() else "cpu"

tokenizer = DistilBertTokenizer.from_pretrained("./distilbert_local")
model          =            DistilBertForSequenceClassification.from_pretrained("./distilbert_local",
num_labels=len(genre_cols)).to(device)

inputs = tokenizer(texts, padding=True, truncation=True, return_tensors="pt", max_length=512)

# Move inputs to GPU if available
inputs = {key: value.to(device) for key, value in inputs.items()}
labels = torch.tensor(labels).to(device)

inputs

### TRAINING

from torch.utils.data import DataLoader, TensorDataset
from torch.optim import AdamW
from torch.optim.lr_scheduler import StepLR

# Prepare data loader
dataset = TensorDataset(inputs['input_ids'], inputs['attention_mask'], labels)
train_dataloader = DataLoader(dataset, batch_size=16, shuffle=True)

# Initialize optimizer
optimizer = AdamW(model.parameters(), lr=4e-5)
class_counts = labels.sum(dim=0)
pos_weights = (len(labels) - class_counts) / (class_counts + 1e-5)
loss_fn   =    torch.nn.BCEWithLogitsLoss(pos_weight=pos_weights) # Handle   multi-label
classification correctly.

# Initialize scheduler
scheduler = StepLR(optimizer, step_size=2, gamma=0.1)  # Adjust the learning rate every 2 epoch

# Early stopping setup
best_loss = float('inf')
patience_counter = 0
patience = 3  # Stop training if no improvement in loss for 2 consecutive epochs

# Start training
model.train()
epochs = 15
for epoch in range(epochs):
    total_loss = 0
    for batch in train_dataloader:
        # Clear gradients
        optimizer.zero_grad()

        # Forward pass
        input_ids, attention_mask, label = batch
        output = model(input_ids, attention_mask=attention_mask)
        loss = loss_fn(output.logits, label.float())

        # Backward pass
        loss.backward()

        # Optimizer step
        optimizer.step()

        total_loss += loss.item()

    avg_loss = total_loss / len(train_dataloader)
    print(f"Epoch {epoch+1}/{epochs} - Loss: {avg_loss:.4f}")

    # Step the scheduler to adjust learning rate
    scheduler.step()

    # Early stopping check
    if avg_loss < best_loss:
        best_loss = avg_loss
        patience_counter = 0  # Reset counter
    else:
        patience_counter += 1
        if patience_counter >= patience:
            print("Early stopping triggered!")
            break

best_loss

model.save_pretrained("./distilbert_finetuned_local")
tokenizer.save_pretrained("./distilbert_finetuned_local")

### PREDICTION

from tqdm import tqdm

predictions = []

with torch.no_grad(): # Disables gradient tracking since we're just predicting.
    for text in tqdm(texts, desc="Predicting genres"):
        encoded =   tokenizer(text,return_tensors='pt',padding=True,truncation=True,
max_length=512).to("cuda")
```

```
            outputs = model(**encoded)
            probs = torch.sigmoid(outputs.logits)
            preds = (probs > 0.5).int().squeeze().tolist()
            predictions.append(preds)

df = pd.DataFrame(predictions, columns=genre_cols)
df = pd.concat([data[["Movie_ID", "Movie_Name"]], df], axis=1)

df.head()

### CLASSIFICATION REPORT

from sklearn.metrics import precision_score, recall_score, f1_score

# Ensure movies and df have the same genre columns
genres_list = [
    "Action", "Adventure", "Animation", "Children's", "Comedy", "Crime",
    "Documentary", "Drama", "Fantasy", "Film Noir", "Horror", "Musical",
    "Mystery", "Romance", "Sci-Fi", "Thriller", "War", "Western"
]

# Initialize result dictionary
results = {"Genre": [], "Precision": [], "Recall": [], "F1-score": []}

# Compute metrics for each genre
for genre in genres_list:
    precision = precision_score(data[genre], df[genre])
    recall = recall_score(data[genre], df[genre])
    f1 = f1_score(data[genre], df[genre])

    # Store results rounded to 2 decimal places
    results["Genre"].append(genre)
    results["Precision"].append(round(precision, 2))
    results["Recall"].append(round(recall, 2))
    results["F1-score"].append(round(f1, 2))

# Convert results into a DataFrame
metrics_df = pd.DataFrame(results)

metrics_df.to_csv("DB_Classification_Score.csv", index=False)

# Display the result
print(metrics_df)


## HYPER-PARAMETER TUNING

!pip install prettytable
```

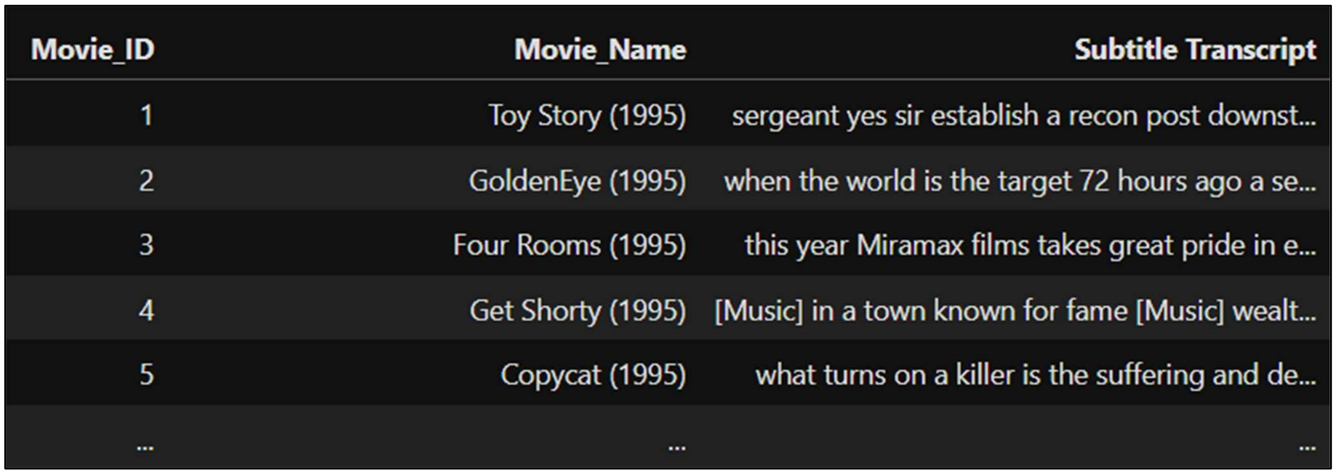
```
!pip install datasets

!pip install optuna

!pip install --upgrade transformers

!pip install accelerate>=0.26.0

!pip install transformers[torch]

pip show accelerate

from sklearn.model_selection import train_test_split
from datasets import Dataset, DatasetDict

device = "cuda" if torch.cuda.is_available() else "cpu"

tokenizer = DistilBertTokenizer.from_pretrained("./distilbert_finetuned_local")
model = DistilBertForSequenceClassification.from_pretrained("./distilbert_finetuned_local",
num_labels=len(genre_cols), problem_type="multi_label_classification").to(device)

# Split the data
train_texts, val_texts, train_labels, val_labels = train_test_split(
    texts, labels, test_size=0.2, random_state=42
)

train_labels = [list(map(float, lbl)) for lbl in train_labels]
val_labels = [list(map(float, lbl)) for lbl in val_labels]

# Tokenize
train_encodings = tokenizer(train_texts, truncation=True, padding=True, max_length=512)
val_encodings = tokenizer(val_texts, truncation=True, padding=True, max_length=512)

# Create HuggingFace datasets
train_dataset = Dataset.from_dict({
    'input_ids': train_encodings['input_ids'],
    'attention_mask': train_encodings['attention_mask'],
    'labels': train_labels
})
val_dataset = Dataset.from_dict({
    'input_ids': val_encodings['input_ids'],
    'attention_mask': val_encodings['attention_mask'],
    'labels': val_labels
})

# Final dictionary
encoded_dataset = DatasetDict({
    'train': train_dataset,
    'validation': val_dataset
})


from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score
import numpy as np
import torch

def compute_metrics(eval_pred):
    logits, labels = eval_pred
    probs = torch.sigmoid(torch.tensor(logits)).numpy()
    preds = (probs > 0.5).astype(int)
    labels = np.array(labels)

    return {
        'precision': precision_score(labels, preds, average='micro', zero_division=0),
        'recall': recall_score(labels, preds, average='micro', zero_division=0),
        'f1': f1_score(labels, preds, average='micro', zero_division=0),
        'accuracy': accuracy_score(labels, preds)
    }

import optuna
from transformers import TrainingArguments, Trainer, EarlyStoppingCallback
from sklearn.metrics import precision_score
import numpy as np

# Objective function for Optuna
def objective(trial):
    # Suggesting hyperparameters
    learning_rate = trial.suggest_float('learning_rate', 1e-5, 5e-5, log=True)  # picks from range
    batch_size = trial.suggest_categorical('batch_size', [8, 16, 32])   # picks from list
    weight_decay = trial.suggest_float('weight_decay', 0.0, 0.3)

    # Define TrainingArguments
    training_args = TrainingArguments(
        output_dir="./optuna_results",
        learning_rate=learning_rate,
        per_device_train_batch_size=batch_size,
        per_device_eval_batch_size=batch_size,
        num_train_epochs=3,
        weight_decay=weight_decay,
        save_strategy="steps",                  # Save model every few steps
        save_steps=100,                                  # Save every 100 steps
        eval_strategy="steps",       # Evaluate every few steps
        eval_steps=100,                                  # Evaluate every 100 steps
        logging_steps=100,                      # Log every 100 steps
        load_best_model_at_end=True,  # Load best model at the end based on evaluation
        disable_tqdm=True,                      # Disable tqdm to avoid overloading output
        report_to="none",                        #    Disable   logging   to    platforms  like
TensorBoard
```

```
)

    # Trainer setup
    trainer = Trainer(
        model=model,
        args=training_args,
        train_dataset=encoded_dataset['train'],
        eval_dataset=encoded_dataset['validation'],
        compute_metrics=compute_metrics,
        callbacks=[EarlyStoppingCallback(early_stopping_patience=2)]
    )

    trainer.train()
    eval_results = trainer.evaluate()
    return eval_results["eval_precision"]  # Maximize precision

# Run Optuna
optuna.logging.set_verbosity(optuna.logging.INFO)

study = optuna.create_study(direction="maximize")
study.optimize(objective, n_trials=10)

print("Best Parameters:", study.best_params)


import json

best_params = {
    "learning_rate": 3.140831723089727e-05,
    "batch_size": 16,
    "weight_decay": 0.2206635125996341
}

#with open("best_hyperparams.json", "w") as f:
#    json.dump(best_params, f)

#with open("best_hyperparams.json", "r") as f:
#    best_params = json.load(f)

from torch.utils.data import DataLoader, TensorDataset
from torch.optim import AdamW
from torch.optim.lr_scheduler import StepLR

inputs = tokenizer(texts, padding=True, truncation=True, return_tensors="pt", max_length=512)

# Move inputs to GPU if available
inputs = {key: value.to(device) for key, value in inputs.items()}
labels = torch.tensor(labels).to(device)
```

```
# Prepare data loader
dataset = TensorDataset(inputs['input_ids'], inputs['attention_mask'], labels)
train_dataloader = DataLoader(dataset, batch_size = best_params['batch_size'], shuffle=True)


# Initialize optimizer
optimizer=  AdamW(model.parameters(), lr =  best_params['learning_rate'], weight_decay =
best_params['weight_decay'])
class_counts = labels.sum(dim=0)
pos_weights = (len(labels) - class_counts) / (class_counts + 1e-5)

loss_fn   =    torch.nn.BCEWithLogitsLoss(pos_weight=pos_weights) # Handle   multi-label
classification correctly.

# Initialize scheduler
scheduler = StepLR(optimizer, step_size=2, gamma=0.1)  # Adjust the learning rate every 2 epoch

# Early stopping setup
best_loss = float('inf')
patience_counter = 0
patience = 3  # Stop training if no improvement in loss for 2 consecutive epochs

# Start training
model.train()
epochs = 15
for epoch in range(epochs):
    total_loss = 0
    for batch in train_dataloader:
        # Clear gradients
        optimizer.zero_grad()

        # Forward pass
        input_ids, attention_mask, label = batch
        output = model(input_ids, attention_mask=attention_mask)
        loss = loss_fn(output.logits, label.float())

        # Backward pass
        loss.backward()

        # Optimizer step
        optimizer.step()

        total_loss += loss.item()

    avg_loss = total_loss / len(train_dataloader)
    print(f"Epoch {epoch+1}/{epochs} - Loss: {avg_loss:.4f}")

    # Step the scheduler to adjust learning rate
    scheduler.step()
    # Early stopping check
    if avg_loss < best_loss:
        best_loss = avg_loss
        patience_counter = 0  # Reset counter
    else:
        patience_counter += 1
        if patience_counter >= patience:
            print("Early stopping triggered!")
            break

best_loss

from tqdm import tqdm

predictions = []

with torch.no_grad(): # Disables gradient tracking since we're just predicting.
    for text in tqdm(texts, desc="Predicting genres"):
        encoded =   tokenizer(text,return_tensors='pt',padding=True,truncation=True,
max_length=512).to("cuda")
        outputs = model(**encoded)
        probs = torch.sigmoid(outputs.logits)
        preds = (probs > 0.5).int().squeeze().tolist()
        predictions.append(preds)

df = pd.DataFrame(predictions, columns=genre_cols)
df = pd.concat([data[["Movie_ID", "Movie_Name"]], df], axis=1)

df.head()

from sklearn.metrics import precision_score, recall_score, f1_score

genres_list = [
    "Action", "Adventure", "Animation", "Children's", "Comedy", "Crime",
    "Documentary", "Drama", "Fantasy", "Film Noir", "Horror", "Musical",
    "Mystery", "Romance", "Sci-Fi", "Thriller", "War", "Western"]

# Initialize result dictionary
results = {"Genre": [], "Precision": [], "Recall": [], "F1-score": []}

# Compute metrics for each genre
for genre in genres_list:
    precision = precision_score(data[genre], df[genre])
    recall = recall_score(data[genre], df[genre])
    f1 = f1_score(data[genre], df[genre])

    # Store results rounded to 2 decimal places
    results["Genre"].append(genre)
    results["Precision"].append(round(precision, 2))
    results["Recall"].append(round(recall, 2))
    results["F1-score"].append(round(f1, 2))

metrics_df = pd.DataFrame(results)

metrics_df.to_csv("DB_Classification_Score.csv", index=False)

# Display the result
print(metrics_df)

model.save_pretrained("./distilbert_HPT_local")
tokenizer.save_pretrained("./distilbert_HPT_local")
```

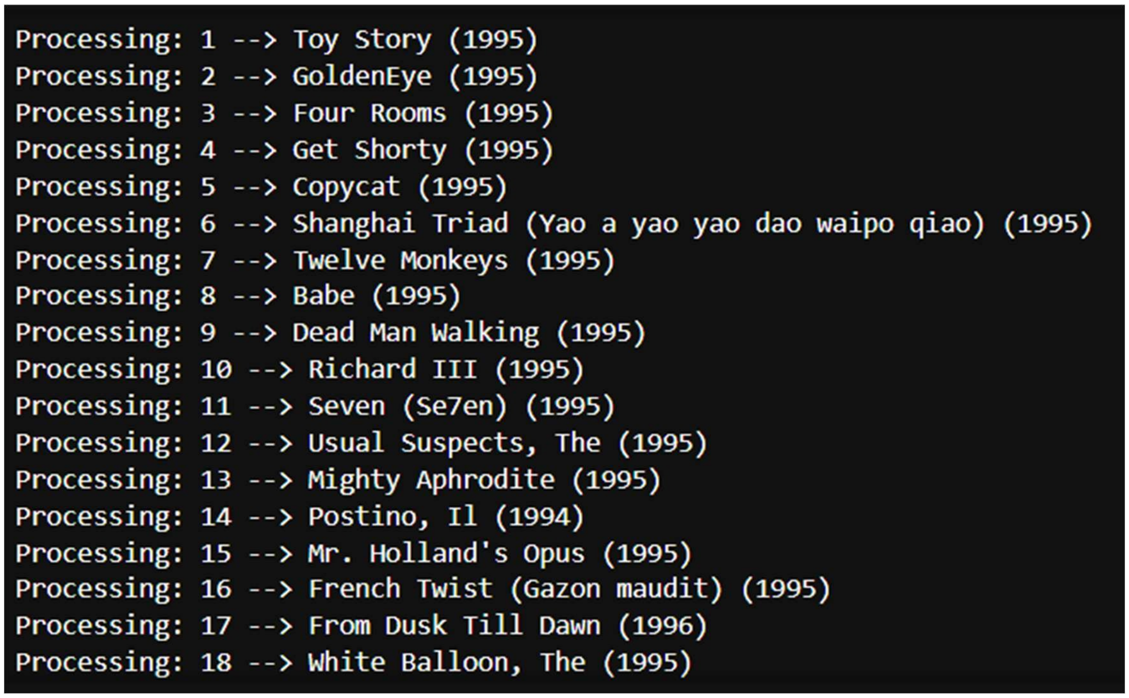
CHAPTER 4

SNAPSHOTS



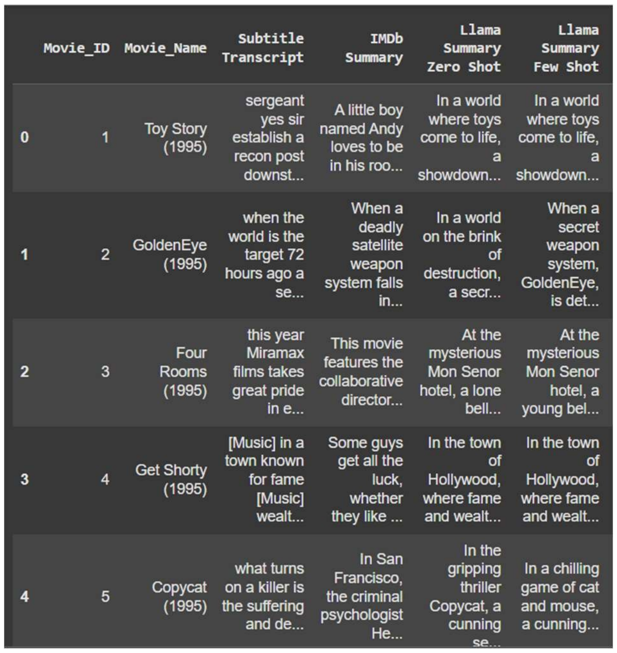
Movie_ID	Movie_Name	Subtitle Transcript
1	Toy Story (1995)	sergeant yes sir establish a recon post downst...
2	GoldenEye (1995)	when the world is the target 72 hours ago a se...
3	Four Rooms (1995)	this year Miramax films takes great pride in e...
4	Get Shorty (1995)	[Music] in a town known for fame [Music] wealt...
5	Copycat (1995)	what turns on a killer is the suffering and de...
...	...	...

Fig. 4.1 – Summary Extraction



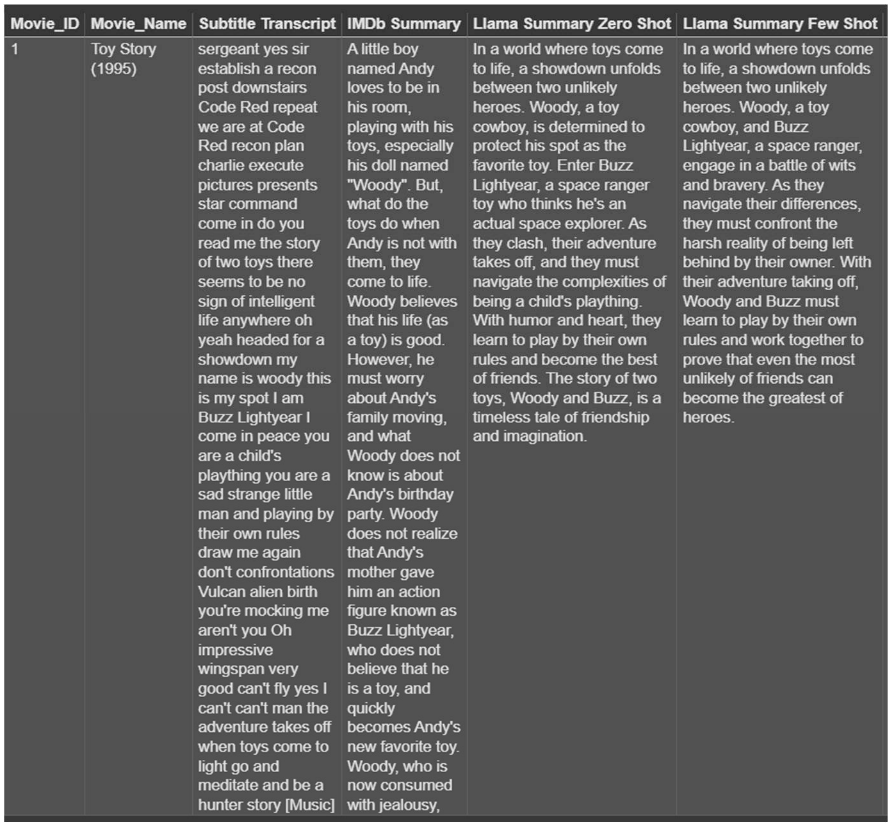
```
Processing: 1 --> Toy Story (1995)
Processing: 2 --> GoldenEye (1995)
Processing: 3 --> Four Rooms (1995)
Processing: 4 --> Get Shorty (1995)
Processing: 5 --> Copycat (1995)
Processing: 6 --> Shanghai Triad (Yao a yao yao dao waipo qiao) (1995)
Processing: 7 --> Twelve Monkeys (1995)
Processing: 8 --> Babe (1995)
Processing: 9 --> Dead Man Walking (1995)
Processing: 10 --> Richard III (1995)
Processing: 11 --> Seven (Se7en) (1995)
Processing: 12 --> Usual Suspects, The (1995)
Processing: 13 --> Mighty Aphrodite (1995)
Processing: 14 --> Postino, Il (1994)
Processing: 15 --> Mr. Holland's Opus (1995)
Processing: 16 --> French Twist (Gazon maudit) (1995)
Processing: 17 --> From Dusk Till Dawn (1996)
Processing: 18 --> White Balloon, The (1995)
```

Fig. 4.2 – Zero Shot Summary Generation



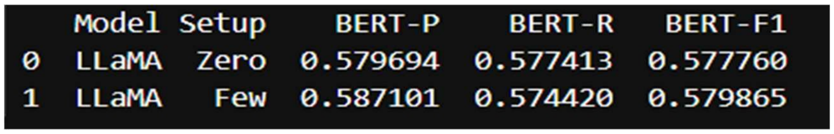
	Movie_ID	Movie_Name	Subtitle Transcript	IMDb Summary	Llama Summary Zero Shot	Llama Summary Few Shot
0	1	Toy Story (1995)	sergeant yes sir establish a recon post downst...	A little boy named Andy loves to be in his roo...	In a world where toys come to life, a showdown...	In a world where toys come to life, a showdown...
1	2	GoldenEye (1995)	when the world is the target 72 hours ago a se...	When a deadly satellite weapon system falls in...	In a world on the brink of destruction, a secr...	When a secret weapon system, GoldenEye, is det...
2	3	Four Rooms (1995)	this year Miramax films takes great pride in e...	This movie features the collaborative director...	At the mysterious Mon Senor hotel, a lone bell...	At the mysterious Mon Senor hotel, a young bel...
3	4	Get Shorty (1995)	[Music] in a town known for fame [Music] wealt...	Some guys get all the luck, whether they like ...	In the town of Hollywood, where fame and wealt...	In the town of Hollywood, where fame and wealt...
4	5	Copycat (1995)	what turns on a killer is the suffering and de...	In San Francisco, the criminal psychologist He...	In the gripping thriller Copycat, a cunning se...	In a chilling game of cat and mouse, a cunning...

Fig. 4.3 – LLM Generated Summaries



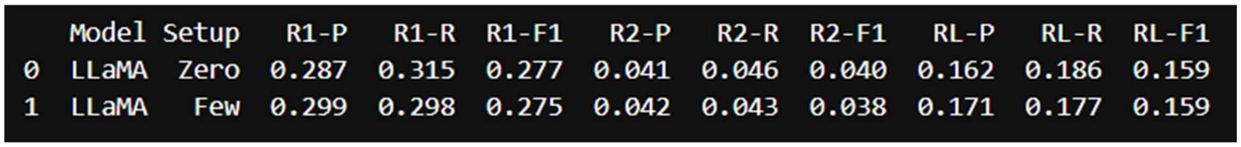
Movie_ID	Movie_Name	Subtitle Transcript	IMDb Summary	Llama Summary Zero Shot	Llama Summary Few Shot
1	Toy Story (1995)	sergeant yes sir establish a recon post downstairs Code Red repeat we are at Code Red recon plan charlie execute pictures presents star command come in do you read me the story of two toys there seems to be no sign of intelligent life anywhere oh yeah headed for a showdown my name is woody this is my spot I am Buzz Lightyear I come in peace you are a child's plaything you are a sad strange little man and playing by their own rules draw me again don't confrontations Vulcan alien birth you're mocking me aren't you Oh impressive wingspan very good can't fly yes I can't can't man the adventure takes off when toys come to light go and meditate and be a hunter story [Music]	A little boy named Andy loves to be in his room, playing with his toys, especially his doll named "Woody". But, what do the toys do when Andy is not with them, they come to life. Woody believes that his life (as a toy) is good. However, he must worry about Andy's family moving, and what Woody does not know is about Andy's birthday party. Woody does not realize that Andy's mother gave him an action figure known as Buzz Lightyear, who does not believe that he is a toy, and quickly becomes Andy's new favorite toy. Woody, who is now consumed with jealousy,	In a world where toys come to life, a showdown unfolds between two unlikely heroes. Woody, a toy cowboy, is determined to protect his spot as the favorite toy. Enter Buzz Lightyear, a space ranger toy who thinks he's an actual space explorer. As they clash, their adventure takes off, and they must navigate the complexities of being a child's plaything. With humor and heart, they learn to play by their own rules and become the best of friends. The story of two toys, Woody and Buzz, is a timeless tale of friendship and imagination.	In a world where toys come to life, a showdown unfolds between two unlikely heroes. Woody, a toy cowboy, and Buzz Lightyear, a space ranger, engage in a battle of wits and bravery. As they navigate their differences, they must confront the harsh reality of being left behind by their owner. With their adventure taking off, Woody and Buzz must learn to play by their own rules and work together to prove that even the most unlikely of friends can become the greatest of heroes.

Fig. 4.4 – Sample Summary Instance



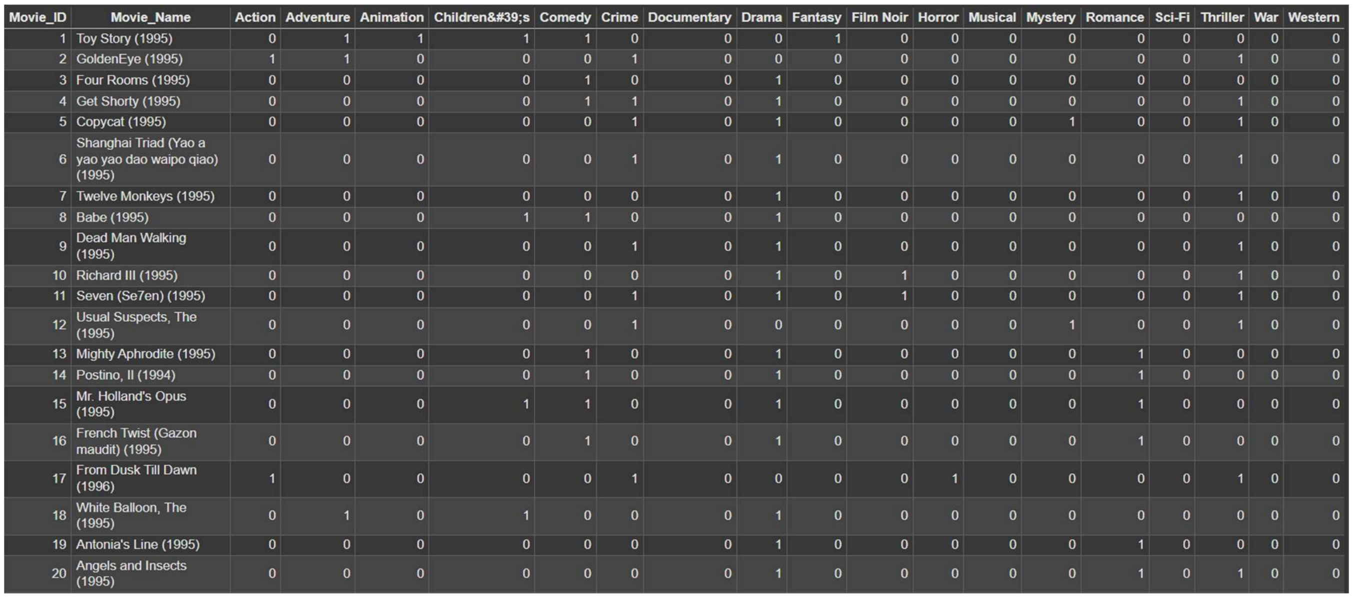
```
   Model Setup    BERT-P    BERT-R   BERT-F1
0  LLaMA  Zero  0.579694  0.577413  0.577760
1  LLaMA   Few  0.587101  0.574420  0.579865
```

Fig. 4.5 – BERT Scores



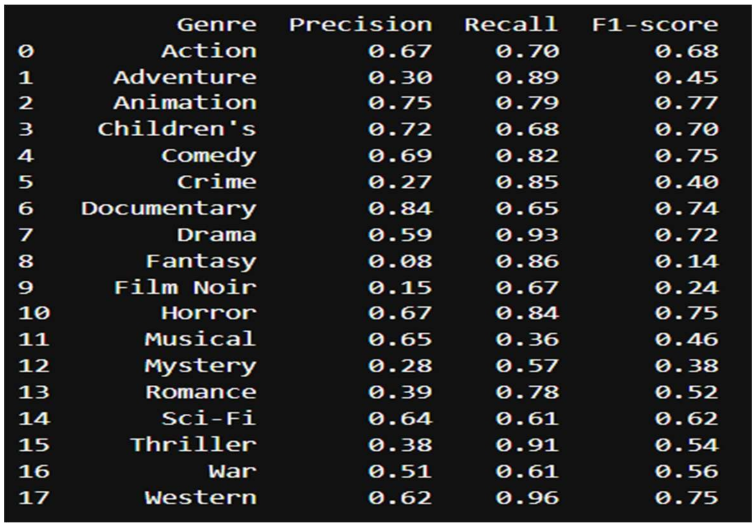
```
   Model Setup   R1-P   R1-R  R1-F1   R2-P   R2-R  R2-F1   RL-P   RL-R  RL-F1
0  LLaMA  Zero  0.287  0.315  0.277  0.041  0.046  0.040  0.162  0.186  0.159
1  LLaMA   Few  0.299  0.298  0.275  0.042  0.043  0.038  0.171  0.177  0.159
```

Fig. 4.6 – ROUGE Metrics



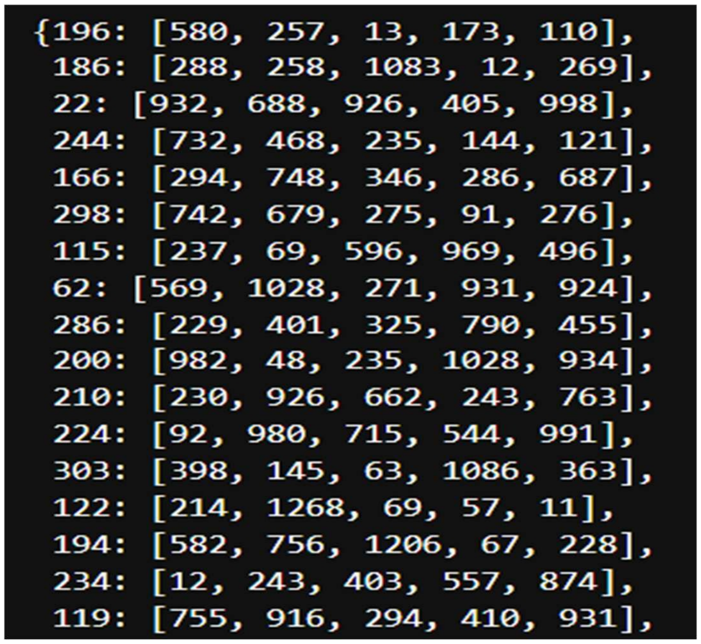
Movie_ID	Movie_Name	Action	Adventure	Animation	Children&#39;s	Comedy	Crime	Documentary	Drama	Fantasy	Film Noir	Horror	Musical	Mystery	Romance	Sci-Fi	Thriller	War	Western
1	Toy Story (1995)	0	1	1	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0
2	GoldenEye (1995)	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0
3	Four Rooms (1995)	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0
4	Get Shorty (1995)	0	0	0	0	1	1	0	1	0	0	0	0	0	0	0	1	0	0
5	Copycat (1995)	0	0	0	0	0	1	0	1	0	0	0	0	1	0	0	1	0	0
6	Shanghai Triad (Yao a yao yao dao waipo qiao) (1995)	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	1	0	0
7	Twelve Monkeys (1995)	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0
8	Babe (1995)	0	0	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0
9	Dead Man Walking (1995)	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	1	0	0
10	Richard III (1995)	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	1	0	0
11	Seven (Se7en) (1995)	0	0	0	0	0	1	0	1	0	1	0	0	0	0	0	1	0	0
12	Usual Suspects, The (1995)	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	1	0	0
13	Mighty Aphrodite (1995)	0	0	0	0	1	0	0	1	0	0	0	0	0	1	0	0	0	0
14	Postino, Il (1994)	0	0	0	0	1	0	0	1	0	0	0	0	0	1	0	0	0	0
15	Mr. Holland's Opus (1995)	0	0	0	1	1	0	0	1	0	0	0	0	0	1	0	0	0	0
16	French Twist (Gazon maudit) (1995)	0	0	0	0	1	0	0	1	0	0	0	0	0	1	0	0	0	0
17	From Dusk Till Dawn (1996)	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	1	0	0
18	White Balloon, The (1995)	0	1	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0
19	Antonia's Line (1995)	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0
20	Angels and Insects (1995)	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	1	0	0

Fig. 4.7 – Genre Prediction



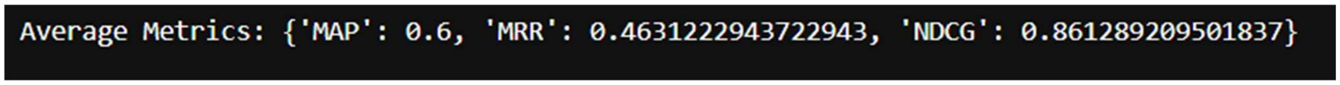
```
          Genre  Precision  Recall  F1-score
0        Action       0.67    0.70      0.68
1     Adventure       0.30    0.89      0.45
2     Animation       0.75    0.79      0.77
3    Children's       0.72    0.68      0.70
4        Comedy       0.69    0.82      0.75
5         Crime       0.27    0.85      0.40
6   Documentary       0.84    0.65      0.74
7         Drama       0.59    0.93      0.72
8       Fantasy       0.08    0.86      0.14
9     Film Noir       0.15    0.67      0.24
10       Horror       0.67    0.84      0.75
11      Musical       0.65    0.36      0.46
12      Mystery       0.28    0.57      0.38
13      Romance       0.39    0.78      0.52
14       Sci-Fi       0.64    0.61      0.62
15     Thriller       0.38    0.91      0.54
16          War       0.51    0.61      0.56
17      Western       0.62    0.96      0.75
```

Fig. 4.8 – Classification Report For Genre Prediction



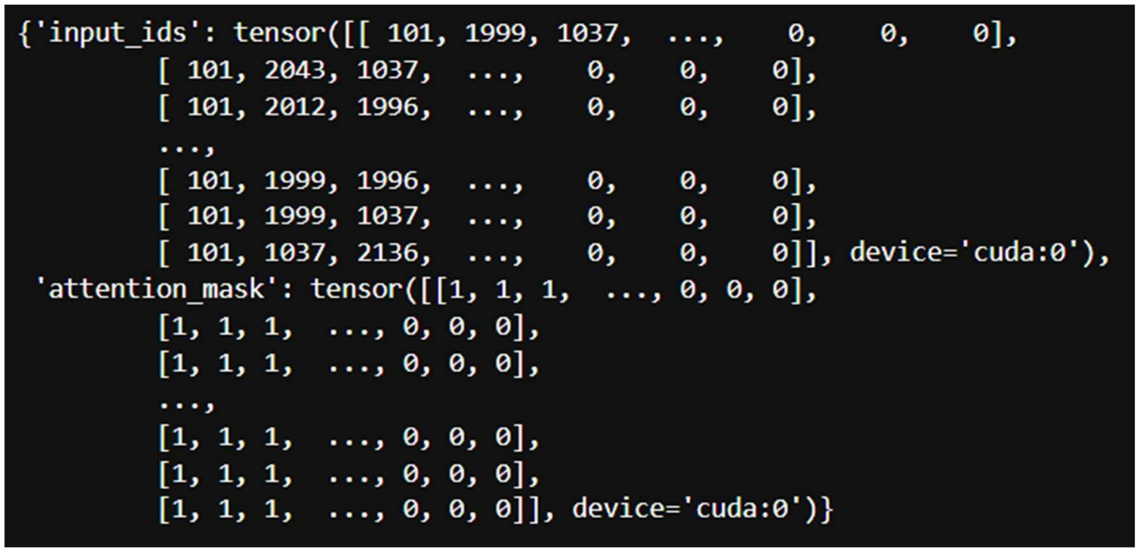
```
{196: [580, 257, 13, 173, 110],
 186: [288, 258, 1083, 12, 269],
 22: [932, 688, 926, 405, 998],
 244: [732, 468, 235, 144, 121],
 166: [294, 748, 346, 286, 687],
 298: [742, 679, 275, 91, 276],
 115: [237, 69, 596, 969, 496],
 62: [569, 1028, 271, 931, 924],
 286: [229, 401, 325, 790, 455],
 200: [982, 48, 235, 1028, 934],
 210: [230, 926, 662, 243, 763],
 224: [92, 980, 715, 544, 991],
 303: [398, 145, 63, 1086, 363],
 122: [214, 1268, 69, 57, 11],
 194: [582, 756, 1206, 67, 228],
 234: [12, 243, 403, 557, 874],
 119: [755, 916, 294, 410, 931],
```

Fig. 4.9 – Ground Truth Rankings



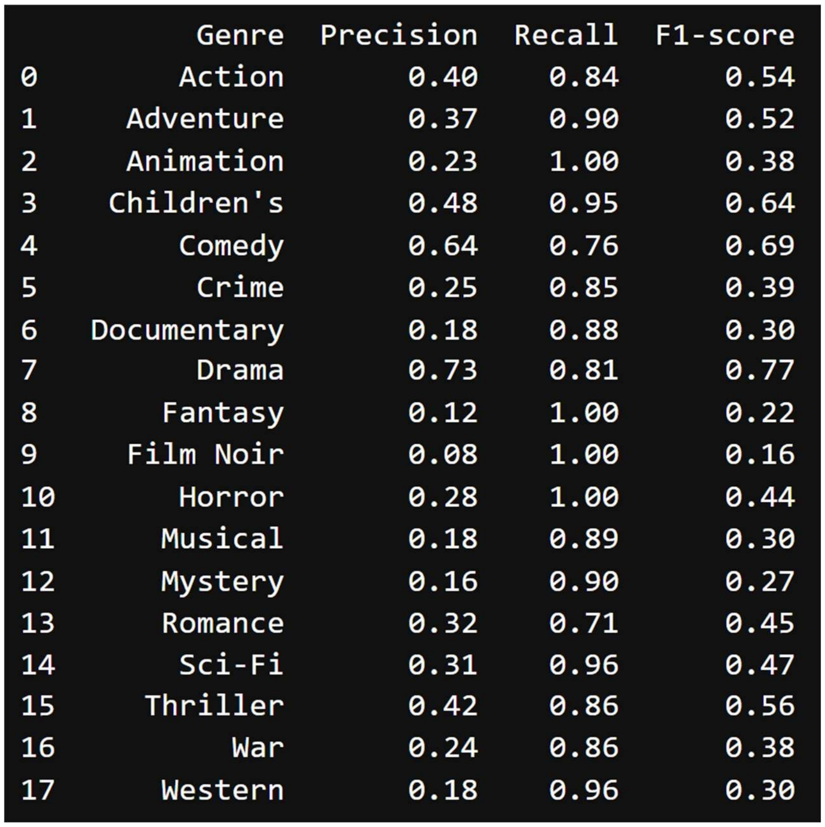
```
Average Metrics: {'MAP': 0.6, 'MRR': 0.4631222943722943, 'NDCG': 0.861289209501837}
```

Fig. 4.10 – MAP, MRR and NDCG Scores



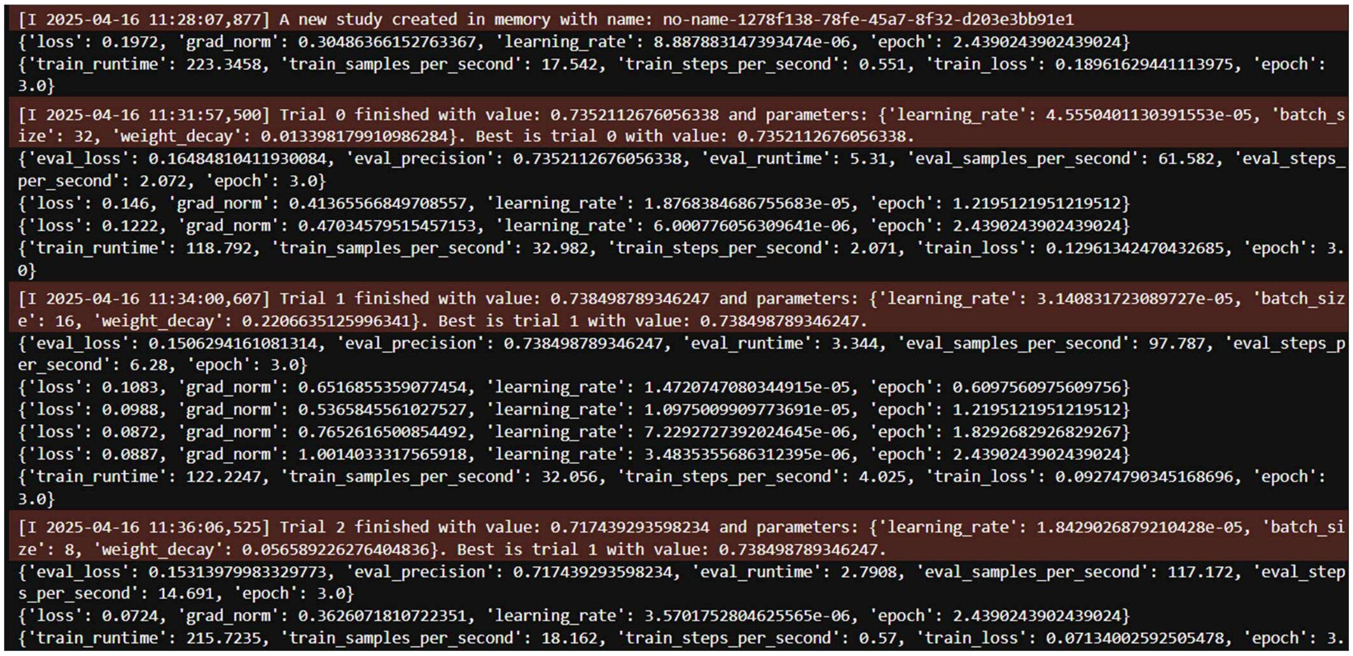
```
{'input_ids': tensor([[ 101, 1999, 1037,  ...,    0,    0,    0],
        [ 101, 2043, 1037,  ...,    0,    0,    0],
        [ 101, 2012, 1996,  ...,    0,    0,    0],
        ...,
        [ 101, 1999, 1996,  ...,    0,    0,    0],
        [ 101, 1999, 1037,  ...,    0,    0,    0],
        [ 101, 1037, 2136,  ...,    0,    0,    0]], device='cuda:0'),
 'attention_mask': tensor([[1, 1, 1,  ..., 0, 0, 0],
        [1, 1, 1,  ..., 0, 0, 0],
        [1, 1, 1,  ..., 0, 0, 0],
        ...,
        [1, 1, 1,  ..., 0, 0, 0],
        [1, 1, 1,  ..., 0, 0, 0],
        [1, 1, 1,  ..., 0, 0, 0]], device='cuda:0')}
```

Fig. 4.11 – Distil-BERT input_ids and attention_masks



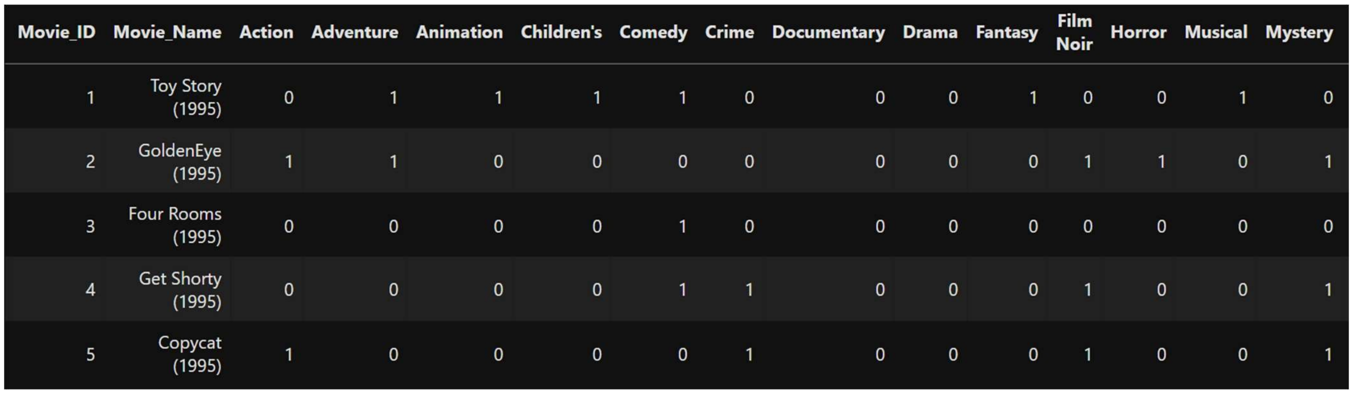
```
             Genre  Precision  Recall  F1-score
0           Action       0.40    0.84      0.54
1        Adventure       0.37    0.90      0.52
2        Animation       0.23    1.00      0.38
3       Children's       0.48    0.95      0.64
4           Comedy       0.64    0.76      0.69
5            Crime       0.25    0.85      0.39
6      Documentary       0.18    0.88      0.30
7            Drama       0.73    0.81      0.77
8          Fantasy       0.12    1.00      0.22
9        Film Noir       0.08    1.00      0.16
10          Horror       0.28    1.00      0.44
11         Musical       0.18    0.89      0.30
12         Mystery       0.16    0.90      0.27
13         Romance       0.32    0.71      0.45
14          Sci-Fi       0.31    0.96      0.47
15        Thriller       0.42    0.86      0.56
16             War       0.24    0.86      0.38
17         Western       0.18    0.96      0.30
```

Fig. 4.12 – Distil-BERT Classification Report
[Before Hyperparameter Tuning]



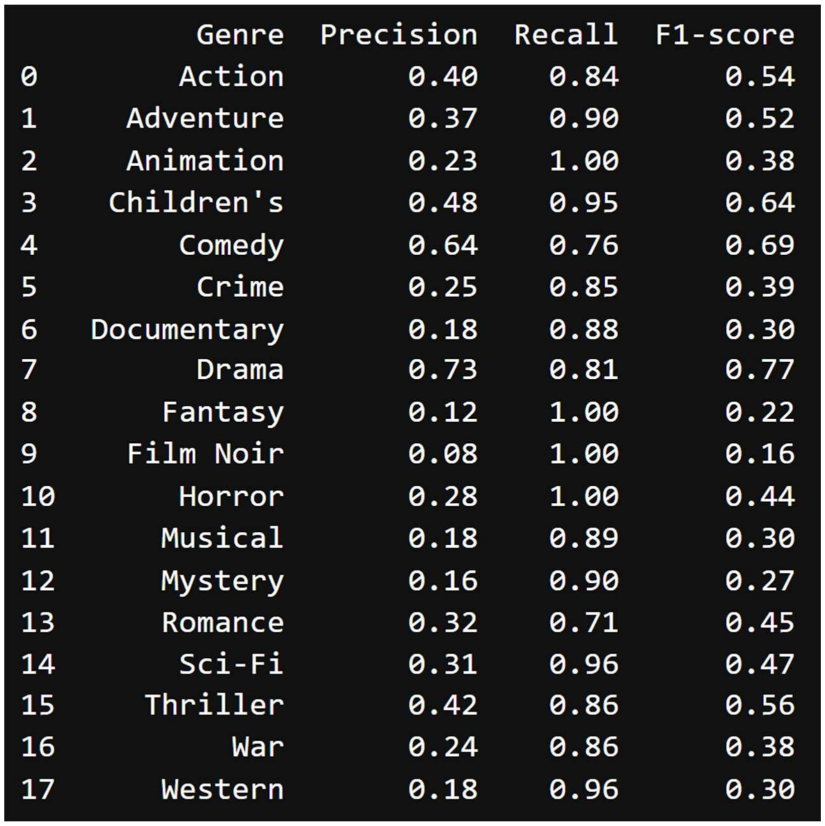
```
[I 2025-04-16 11:28:07,877] A new study created in memory with name: no-name-1278f138-78fe-45a7-8f32-d203e3bb91e1
{'loss': 0.1972, 'grad_norm': 0.30486366152763367, 'learning_rate': 8.887883147393474e-06, 'epoch': 2.4390243902439024}
{'train_runtime': 223.3458, 'train_samples_per_second': 17.542, 'train_steps_per_second': 0.551, 'train_loss': 0.18961629441113975, 'epoch':
3.0}
[I 2025-04-16 11:31:57,500] Trial 0 finished with value: 0.7352112676056338 and parameters: {'learning_rate': 4.5550401130391553e-05, 'batch_s
ize': 32, 'weight_decay': 0.013398179910986284}. Best is trial 0 with value: 0.7352112676056338.
{'eval_loss': 0.16484810411930084, 'eval_precision': 0.7352112676056338, 'eval_runtime': 5.31, 'eval_samples_per_second': 61.582, 'eval_steps_
per_second': 2.072, 'epoch': 3.0}
{'loss': 0.146, 'grad_norm': 0.41365566849708557, 'learning_rate': 1.8768384686755683e-05, 'epoch': 1.2195121951219512}
{'loss': 0.1222, 'grad_norm': 0.47034579515457153, 'learning_rate': 6.000776056309641e-06, 'epoch': 2.4390243902439024}
{'train_runtime': 118.792, 'train_samples_per_second': 32.982, 'train_steps_per_second': 2.071, 'train_loss': 0.12961342470432685, 'epoch': 3.
0}
[I 2025-04-16 11:34:00,607] Trial 1 finished with value: 0.738498789346247 and parameters: {'learning_rate': 3.140831723089727e-05, 'batch_siz
e': 16, 'weight_decay': 0.2206635125996341}. Best is trial 1 with value: 0.738498789346247.
{'eval_loss': 0.1506294161081314, 'eval_precision': 0.738498789346247, 'eval_runtime': 3.344, 'eval_samples_per_second': 97.787, 'eval_steps_p
er_second': 6.28, 'epoch': 3.0}
{'loss': 0.1083, 'grad_norm': 0.6516855359077454, 'learning_rate': 1.4720747080344915e-05, 'epoch': 0.6097560975609756}
{'loss': 0.0988, 'grad_norm': 0.5365845561027527, 'learning_rate': 1.0975009909773691e-05, 'epoch': 1.2195121951219512}
{'loss': 0.0872, 'grad_norm': 0.7652616500854492, 'learning_rate': 7.2292727392024645e-06, 'epoch': 1.8292682926829267}
{'loss': 0.0887, 'grad_norm': 1.0014033317565918, 'learning_rate': 3.4835355686312395e-06, 'epoch': 2.4390243902439024}
{'train_runtime': 122.2247, 'train_samples_per_second': 32.056, 'train_steps_per_second': 4.025, 'train_loss': 0.09274790345168696, 'epoch':
3.0}
[I 2025-04-16 11:36:06,525] Trial 2 finished with value: 0.717439293598234 and parameters: {'learning_rate': 1.8429026879210428e-05, 'batch_si
ze': 8, 'weight_decay': 0.056589226276404836}. Best is trial 1 with value: 0.738498789346247.
{'eval_loss': 0.15313979983329773, 'eval_precision': 0.717439293598234, 'eval_runtime': 2.7908, 'eval_samples_per_second': 117.172, 'eval_step
s_per_second': 14.691, 'epoch': 3.0}
{'loss': 0.0724, 'grad_norm': 0.3626071810722351, 'learning_rate': 3.5701752804625565e-06, 'epoch': 2.4390243902439024}
{'train_runtime': 215.7235, 'train_samples_per_second': 18.162, 'train_steps_per_second': 0.57, 'train_loss': 0.07134002592505478, 'epoch': 3.
```

Fig. 4.13 – Distil-BERT Hyperparameter Tuning



Movie_ID	Movie_Name	Action	Adventure	Animation	Children's	Comedy	Crime	Documentary	Drama	Fantasy	Film Noir	Horror	Musical	Mystery
1	Toy Story (1995)	0	1	1	1	1	0	0	0	1	0	0	1	0
2	GoldenEye (1995)	1	1	0	0	0	0	0	0	0	1	1	0	1
3	Four Rooms (1995)	0	0	0	0	1	0	0	0	0	0	0	0	0
4	Get Shorty (1995)	0	0	0	0	1	1	0	0	0	1	0	0	1
5	Copycat (1995)	1	0	0	0	0	1	0	0	0	1	0	0	1

Fig. 4.14 – Distil-BERT Genre Prediction



```
          Genre  Precision  Recall  F1-score
0        Action       0.40    0.84      0.54
1     Adventure       0.37    0.90      0.52
2     Animation       0.23    1.00      0.38
3    Children's       0.48    0.95      0.64
4        Comedy       0.64    0.76      0.69
5         Crime       0.25    0.85      0.39
6   Documentary       0.18    0.88      0.30
7         Drama       0.73    0.81      0.77
8       Fantasy       0.12    1.00      0.22
9     Film Noir       0.08    1.00      0.16
10       Horror       0.28    1.00      0.44
11      Musical       0.18    0.89      0.30
12      Mystery       0.16    0.90      0.27
13      Romance       0.32    0.71      0.45
14       Sci-Fi       0.31    0.96      0.47
15     Thriller       0.42    0.86      0.56
16          War       0.24    0.86      0.38
17      Western       0.18    0.96      0.30
```

Fig. 4.15 – Distil-BERT Classification Report
[After Hyperparameter Tuning]

CHAPTER 5

CONCLUSIONS AND FUTURE PLANS

The mini-project successfully demonstrated the efficacy of large language models (LLMs) li
Llama-3.1-8b and Distil-BERT in enhancing movie recommendation and genre classification by
leveraging movie trailer subtitles, outperforming traditional methods like collaborative filtering o
the MovieLens-100K dataset. The LLM-based approach effectively addressed the cold-start
problem and data sparsity, achieving high precision, recall, and F1-scores in genre prediction, an
superior MAP, MRR, and NDCG scores in recommendations. Distil-BERT's hyperparameter
tuning further improved genre classification accuracy, showcasing the value of model optimizatio
Future work includes scaling the system to larger datasets like MovieLens-1M, integrating
multimodal data (e.g., visual trailer features), optimizing computational efficiency via model
distillation, and developing real-time user feedback mechanisms to enhance personalization an
deployment feasibility.

CHAPTER 6

REFERENCES

[1] S. Raj, P. Mondal, D. Chakder, S. Saha, and N. Onoe (2023), A multi-modalmulti-task base approach for movie recommendation. Proceedings of the International Conference on Neu Networks, vol. 7, pp. 1–8, Jun. 2023.

[2] D. Chakder, P. Mondal, S. Raj, S. Saha, A. Ghosh, and N. Onoe (2022), Graph network based approaches for multi-modal movie recommendation system. Proceedings of the IEEE Internation Conference on Systems, Man, and Cybernetics (SMC), pp. 409–414, Oct. 2022.

[3] S. Pingali, P. Mondal, D. Chakder, S. Saha, and A. Ghosh (2022), Towards developing a multi-modal video recommendation system. Proceedings of the International Joint Conference on Neu Networks (IJCNN), pp. 1–8, Jul. 2022.

[4] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl (2004), Evaluating collaborative filtering recommender systems ACM Transactions on Information Systems, vol. 22, no. 1, pp. 5–53, Jan. 2004.

[5] Y. Koren, R. Bell, and C. Volinsky (2009), Matrix factorization techniques for recommender systems. Computer, vol. 42, no. 8, pp. 30–37, Aug. 2009.

[6] R. Van Meteren and M. Van Someren (2000), Using content-based filtering for recommendation. Proceedings of the MLnet/ECML Workshop on Machine Learning in the New Information Age, vol. 30, 2000, pp. 47–56.

CHAPTER 7

APPENDIX - BASE PAPER

TITLE:

“Transformative Movie Discovery: Large Language Models for Recommendation and Genre Prediction”

CITATION:

S. Raj, A. Sharma, S. Saha, B. Singh, and N. Pedanekar, "Transformative Movie Discovery: Large Language Models for Recommendation and Genre Prediction", IEEE Access, vol. 12, pp. 186626–186637, 2024. (https://ieeexplore.ieee.org/document/10720773)