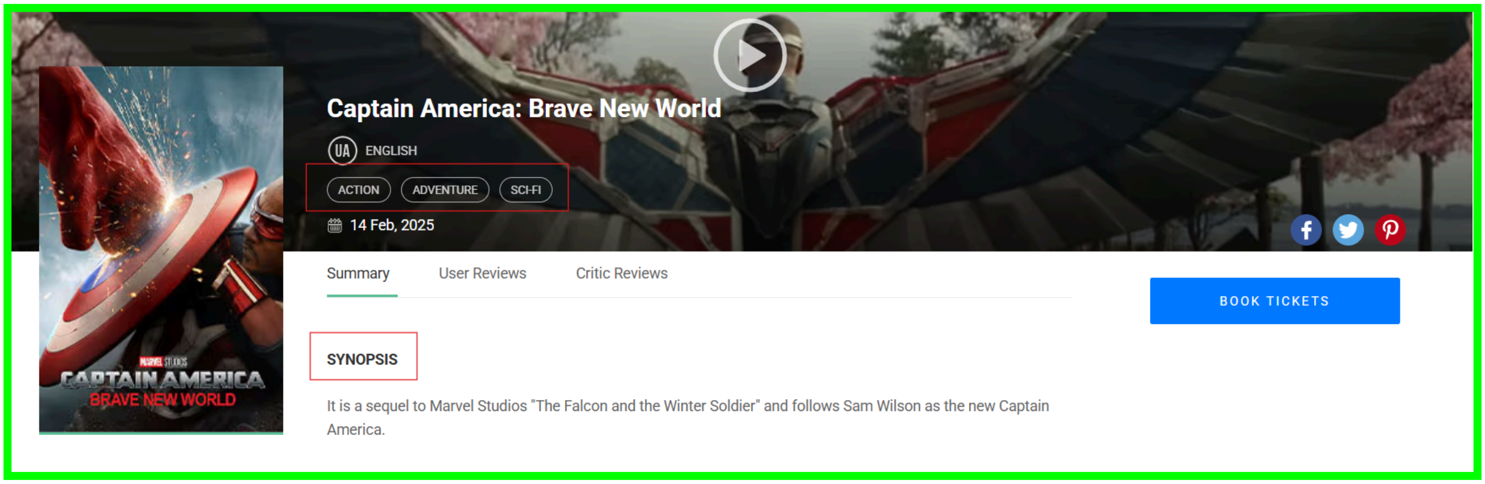
🎬 MOVIE GURNER REVEL BOT 🤖

Introduction

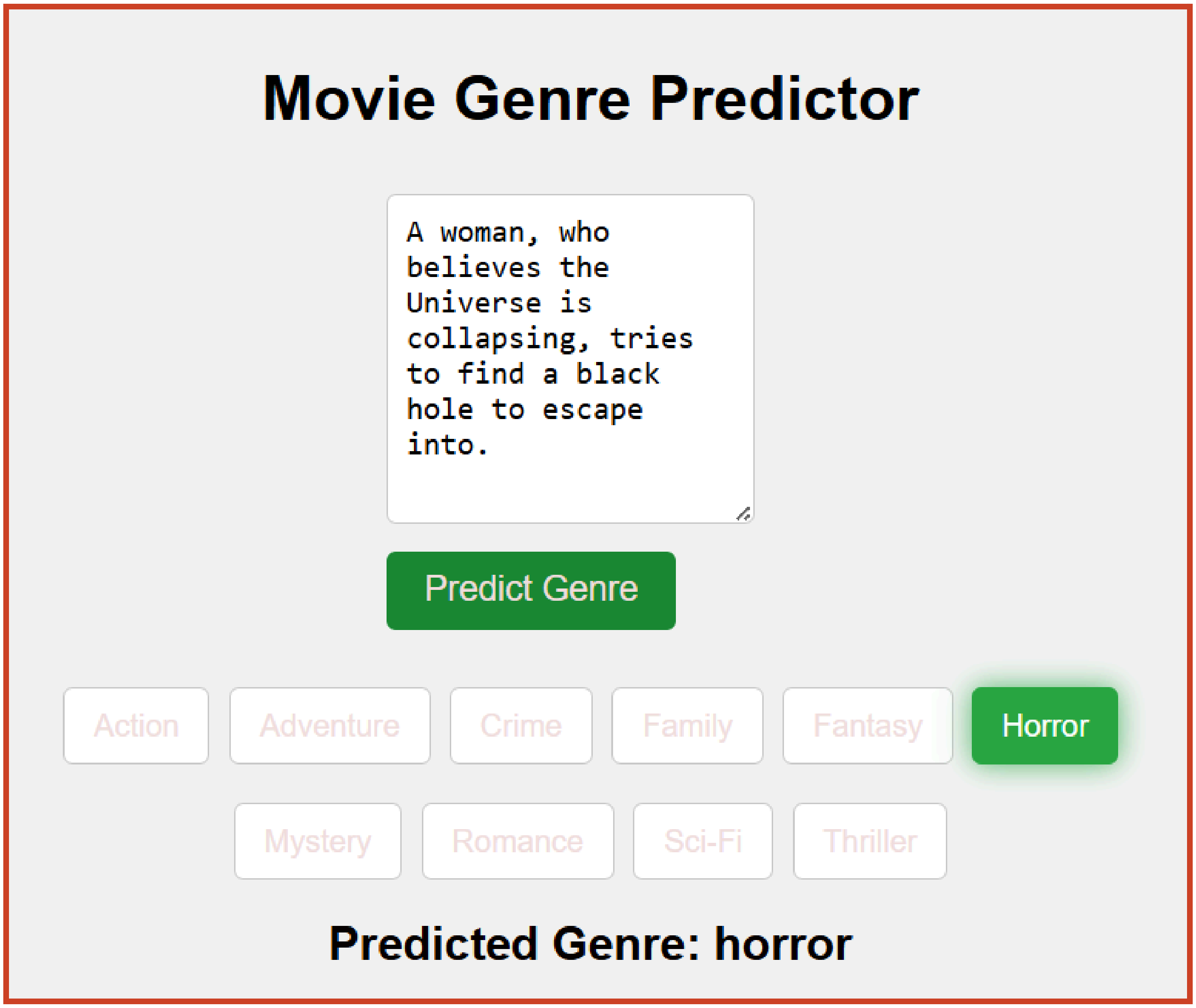
On the BookMyShow website, everyone can see the movie synopsis and genre below the picture.



I plan to automate this system so that when a movie synopsis is provided as input, the bot can accurately identify its genre. Beyond genre prediction, this system can be applied to various real-world scenarios, including:

- Predicting User Sentiment Based on Reviews: Analyzing audience feedback to determine overall sentiment and satisfaction.
- Recommending Similar Movies: Suggesting films based on genre, themes, and storytelling style to enhance user experience.
- Enhancing Streaming Platform Recommendation Engines: Improving content discovery by integrating genre-based recommendations.
- Providing Script Insights: Analyzing scripts to ensure genre consistency and offering suggestions for improvement.

Out bot interface



- Input: Type the movie synopsis.
- Output: What genre does that movie belong to?

Installment details

1. Create a Virtual Environment with Python 3.10

To create a new virtual environment using Python 3.10, run the command to set up the environment.

```
conda create -n cardezzdev python==3.10
```

2. Activate the Conda Environment

Once the environment is created, activate it to start using the packages within this isolated setup.

```
conda activate  cardezzdev
```

3. Install Necessary Libraries and Run Setup

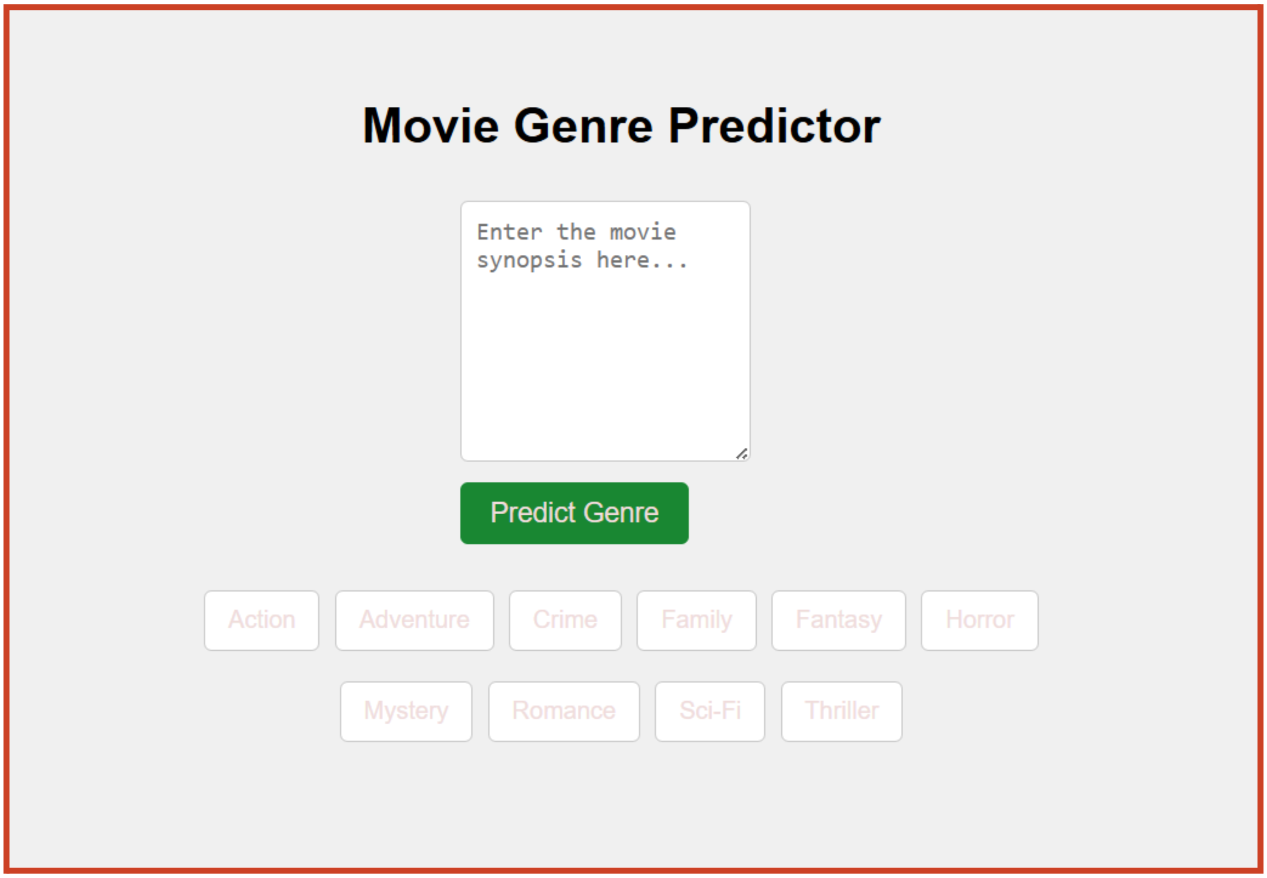
After activating the environment, install the required libraries by running the setup script. This will configure the necessary dependencies for the application.

```
python setup.py
```

If you run setup.py, you will get…

```
(cdazzdev) C:\Users\THUVAA\Desktop\llm-finetuning\book_my_show_movie_genre_Identification
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a pr
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 905-507-677
127.0.0.1 - - [07/Mar/2025 07:24:41] "GET / HTTP/1.1" 200 -
```

Run the local host http://127.0.0.1:5000



If you type the synopsis and you got the output

How to Do This?

Dataset

To achieve this, we first need a dataset. We can accomplish this in two ways:

- Dataset Generation: The first way is by generating the dataset using the prompt engineering techniques provided in the mshumer/gpt-llm-trainer.
- Using Open Datasets: The second way is to find suitable datasets from open resources, which is often the best approach for real-time scenarios.

Sample dataset

	synopsis	genre
0	After his brother is killed and father severel...	action
1	Forced for some time to be a fighting slave, a...	action
2	A young couple must overcome tremendous odds t...	adventure
3	A young girl has been left an orphan but in th...	family
4	A family is cursed Morgann who keeps coming ba...	horror

We obtained a large dataset from Kaggle, specifically the movie_genre_kaggle.csv with more than 50,000 rows. However, for fine-tuning purposes, we only needed a smaller subset of data. So, we sorted the data and created a balanced dataset, genre_counts.csv, with 80 rows, consisting of 10 different genres. Each genre has 8 samples.

- Check the Dataset folder to see if the datasets are available.
- Check the Data_Analysis.py for the conversion from big dataset to sorted dataset

Model Selection

For this task, I can use two models for fine-tuning: GPT-3.5 Turbo and LLaMA 2 7B. There are some key considerations to keep in mind. First, resource availability—if we choose LLaMA 2 7B, we need GPUs for fine-tuning using parameter-efficient fine-tuning (PEFT). Google Colab provides a **T4 GPU** with **12GB RAM** for a limited time. However, adjusting certain PEFT input parameters can increase memory (RAM) consumption, making it difficult to test all configurations on a T4. To overcome this, we can also use Kaggle's **A100 GPU**, which provides more memory.Additionally, structuring the codebase efficiently is challenging due to resource constraints. On the other hand, using GPT-3.5 Turbo requires an API key from OpenAI, which involves a cost. Considering these factors, I chose GPT-3.5 Turbo for this task.

Now, I have the dataset stored in Drive and have chosen the model.

Model Fine Tuning

MovieGenreTrainer ⇒ gpt_finetune_trainer.py

The MovieGenreTrainer class is designed to fine-tune the OpenAI GPT-3.5 Turbo model for movie genre classification based on movie synopsis.This class automates the process of preparing data, generating prompts, and fine-tuning GPT-3.5 Turbo for movie genre classification.

Train ⇒ gpt_train.py

Input

⇒CSV Dataset (genre_counts.csv) → Contains movie synopses and their genres.
⇒API Key → Required to access OpenAI's fine-tuning services.
⇒System Prompt → Defines the model's task.

Output

⇒Training & Inference JSONL files → Processed dataset saved in out/training_examples.jsonl & out/inference_examples.jsonl.
⇒Fine-Tuned Model Name → Generated after fine-tuning is complete.
⇒Fine-Tuning Status Logs → Displays the progress of the training process.

If you run `train.py`, you will get the fine-tuned model.

How to identify if the model was created?

This is an event of the finetuning

```
(cdazzdev) C:\Users\THUVAA\Desktop\llm-finetuning\book_my_show_movie_genre_Identification\gpt_3.5>python gpt_train.py
```


```
{
  "object": "list",
  "data": [
    {
      "object": "fine_tuning.job.event",
      "id": "ftevent-UG1MLLODFqebHhPtGeboUeof",
      "created_at": 1741276402,
      "level": "info",
      "message": "Validating training file: file-D6veHk9ddsJUpc8TrBpqfH",
      "data": {},
      "type": "message"
    },
    {
      "object": "fine_tuning.job.event",
      "id": "ftevent-DZpvA6XcZkwe98iAS5NXsC57",
      "created_at": 1741276402,
      "level": "info",
      "message": "Created fine-tuning job: ftjob-ej0OXURxI7feWr8IK7vCiQbb",
      "data": {},
      "type": "message"
    }
  ],
  "has_more": false
}
None
```


Box one contains the Job ID.Using the Job ID, we can check the fine-tuned job status.The status has four stages: first, file validation; then queued; followed by running; and finally, either succeeded or failed.

```
Job status: validating_files
Job status: validating_files
Job status: queued
Job status: queued
Job status: running
Job status: running
Fine-tuning job finished with status: succeeded
```

And also got the if your model were successfully deployed

OpenAI

Your fine-tuning job `ftjob` has successfully completed, and a new model `ft:gpt-3.5-turbo-0125:personal:` has been created for your use.

Inference

This script defines a MovieGenrePredictor class that uses OpenAI's API to predict a movie's genre based on its synopsis. The class is initialized with the API key, the fine-tuned model, a system message that provides genre classification instructions, and an optional temperature parameter to control response randomness.

The generate_response method sends the movie synopsis to the API, which returns the predicted genre (e.g., 'action,' 'romance,' 'scifi', ect.). The script loads the API key from an .env.example file and demonstrates how the model predicts a genre for a given movie synopsis.

Finally, you get the accurate answers.

Interface

app.py ⇒ Use this Flask API to run the bot on your local host, and you will also get the previous interface.

Accuracy

To evaluate the accuracy of the multi-class classification model, we use several key metrics.

- Confusion Matrix: To visualize the distribution of true and false predictions across all classes.
- Precision: To measure the proportion of correctly predicted positive instances for each class.
- Recall: To assess the proportion of actual positive instances that were correctly identified by the model.
- Weighted Average: To compute the average of metrics, accounting for class imbalances.
- ROC Curve and AUC: To evaluate the model's ability to distinguish between classes, with AUC indicating the overall performance.