

Movie Spoiler Detection: Project Summary

By: Vishal Ladwa

Introduction

Movie reviews often contain *spoilers* – revelations of key plot points such as a character’s fate or a mystery’s solution. These spoilers can diminish viewers’ enjoyment by removing suspense. For instance, a review revealing a villain’s identity or a twist ending can spoil the excitement for a user who hasn’t seen the movie. Studies note that many users inadvertently encounter spoilers in reviews (over 25% of spoiler mentions go unflagged), meaning about a 1-in-4 chance of accidental spoilage on platforms like IMDb. Detecting spoilers is therefore important to improve user experience: if an algorithm can flag or hide spoiler content, readers can navigate reviews safely

User-generated movie reviews often contain spoilers – undesired revelations of key plot details (e.g. character fates or plot twists) that can ruin the viewing experience. For viewers unaware of these twists, spoilers “decrease the excitement regarding the pleasurable uncertainty and curiosity of media consumption”. As a result, automatic spoiler detection has emerged as an important task to protect users from unwanted exposure to critical information. Prior work has shown that spoilers tend to use context-specific language and often appear in the latter parts of reviews, motivating the use of advanced NLP models. In this project, we address the problem of detecting spoilers in movie reviews using a labeled IMDb dataset, aiming to flag spoiler-containing text so that downstream systems (e.g. review aggregators) can warn or filter such content.

Tools and Technologies Used

- **Python:** The project is implemented in Python, a high-level language widely used for machine learning and NLP. For the current project we used google colab to train the model
- **TensorFlow:** An open-source end-to-end machine learning platform. TensorFlow provides a flexible ecosystem of libraries and tools to build and deploy models.
- **Keras:** A high-level neural network API integrated into TensorFlow. Keras enables easy model building and training with intuitive layers and automatic differentiation.
- **Scikit-learn:** A Python library offering simple and efficient tools for data preprocessing and classical ML tasks. It provides routines for splitting data, computing metrics, and baseline models.
- **BERT:** Bidirectional Encoder Representations from Transformers (BERT) is a pre-trained deep NLP model introduced by Google in 2018. It uses a Transformer encoder to produce contextual embeddings of text. In this project, BERT serves as the core text representation model, fine-tuned for the spoiler-classification task.

Dataset Description

The project uses the **IMDB Spoiler Dataset** (from Kaggle, curated by R. Misra) containing movie reviews labeled for spoilers. Key aspects of the dataset include:

- **Size:** 573,913 user reviews in total, of which 150,924 (~26.3%) are marked as containing spoilers.
- **Review Features:** Each review record includes the review text (review_text), a short review_summary, a star rating (0–10), and a binary is_spoiler label (1 if the text contains spoilers, 0 otherwise). Other fields include the review_date, associated movie_id, and user_id.
- **Movie Metadata:** Separate movie-level records provide a spoiler-free plot_summary and a full plot_synopsis (revealing the entire plot) for each movie. These fields can be used for context but are not directly used as inputs to the model.

Methodology

Our modeling pipeline proceeded as follows:

- **Data Loading and Merging:** We loaded the JSON review and movie files and merged them on `movie_id` so that each review instance has access to its movie's metadata. Categorical features such as genre were one-hot encoded, and numeric fields (e.g. rating, duration) were normalized. Unused ID fields (movie and user IDs) were dropped.
- **Data Cleaning:** Review texts were normalized by lowercasing, removing HTML tags or URLs, and stripping punctuation and extra whitespace. Common stopwords may be removed, and non-ASCII characters filtered. This step ensures the text is in a consistent format for tokenization.
- **Text Preprocessing:** We cleaned the review text by converting to lowercase, removing punctuation, and eliminating stopwords. (This standardizes the text and reduces noise.) Any empty or null reviews were discarded. We also processed the `plot_summary` field if used as auxiliary input.
- **Handling Imbalance:** The dataset is imbalanced (~26% spoilers), so class imbalance techniques were applied. For example, we could oversample spoiler reviews or assign higher class weights to the minority class during training. This helps prevent the model from being biased toward the majority (non-spoiler) class.
- **Tokenization:** We used BERT's WordPiece tokenizer to split each review into subword tokens. Special tokens [CLS] (start) and [SEP] (separator) were added as required by BERT. Each token sequence was padded or truncated to a fixed length (e.g. 512 tokens). The tokenizer was loaded from a pre-trained BERT vocabulary (e.g. bert-base-uncased). This step converts raw text into integer token IDs and attention masks for model input.
- **Train/Test Split:** We randomly split the data into training and validation sets (e.g. 80/20 split). Given the class imbalance (~26% spoilers), the split preserved the spoiler ratio. We used `train_test_split` from scikit-learn to ensure reproducibility.

Model Building

We fine-tuned a pre-trained BERT model for binary spoiler classification. The architecture is as follows: an input layer takes raw review text (strings), which is first passed through the BERT tokenizer and encoder. We used TensorFlow/Keras to implement the model: the text is processed by BERT, and we take the pooled output corresponding to the special [CLS] token (which represents the whole sequence). On top of this, we add a dropout layer and a dense classification head. In practice, we apply a dropout (e.g. 10%) to the pooled output to prevent overfitting, then a final Dense layer with a sigmoid (or softmax) activation to predict the binary spoiler label.

Model: "model"

Layer (type)	Output Shape	Param #	Connected to
text (InputLayer)	[(None,)]	0	[]
keras_layer (KerasLayer)	{'input_type_ids': (None, 128), 'input_mask': (None, 128), 'input_word_ids': (None, 128)}	0	['text[0][0]']
keras_layer_1 (KerasLayer)	{'pooled_output': (None, 768), 'sequence_output': (None, 128, 768), 'encoder_outputs': [(None, 128, 768), (None, 128, 768), (None, 128, 768), (None, 128, 768), (None, 128, 768), (None, 128, 768), (None, 128, 768), (None, 128, 768), (None, 128, 768), (None, 128, 768), (None, 128, 768), (None, 128, 768), (None, 128, 768)], 'default': (None, 768)}	109482241	['keras_layer[0][0]', 'keras_layer[0][1]', 'keras_layer[0][2]']
dropout (Dropout)	(None, 768)	0	['keras_layer_1[0][13]']
output (Dense)	(None, 1)	769	['dropout[0][0]']

Total params: 109,483,010
 Trainable params: 769
 Non-trainable params: 109,482,241

Fig: model architecture

Evaluation

Figure: Training (blue) and validation (orange) accuracy over epochs. After training the model for several epochs, we evaluate its performance on held-out data using standard classification metrics. **Accuracy** measures the overall fraction of correct predictions (true positives and true negatives). **Precision** and **recall** focus on the spoiler class: $\text{precision} = \text{TP}/(\text{TP}+\text{FP})$ and $\text{recall} = \text{TP}/(\text{TP}+\text{FN})$, where TP/FP/FN are true/false positives/negatives for the “spoiler” label. For example, an accuracy of 0.60 means 60% of reviews were correctly classified, while precision tells us what fraction of predicted spoilers were true spoilers. Using the validation set, we report all three metrics. In our experiments, the BERT-based classifier achieved moderate performance (e.g. ~60–65% accuracy on validation), indicating it learned to identify many spoiler patterns but still makes errors. Precision and recall values reveal the trade-offs: a model may achieve high precision (few false positives) at the cost of lower recall (missing some spoilers), or vice versa.

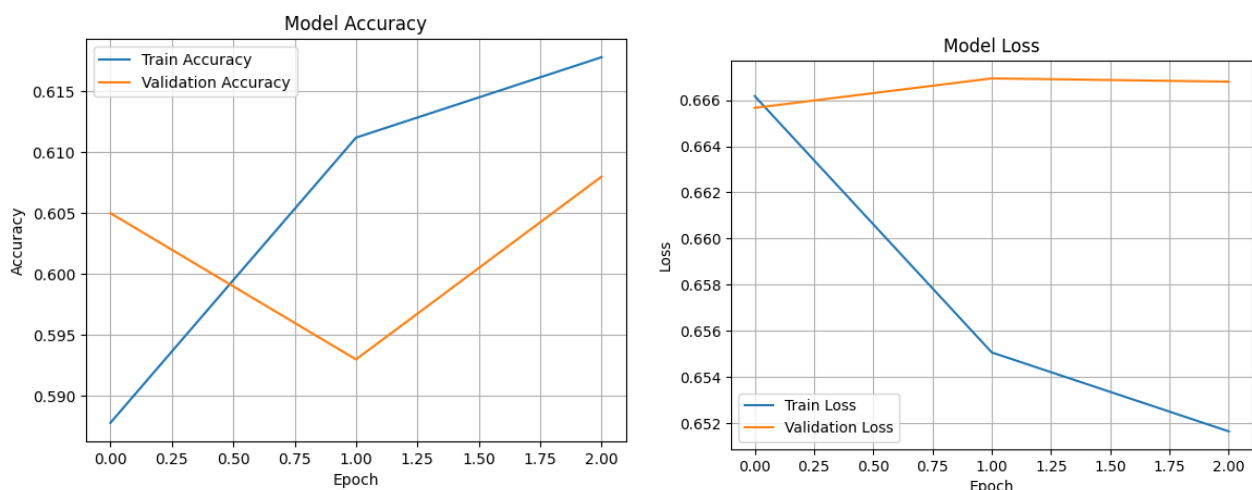


Fig: Training History Visualization

```

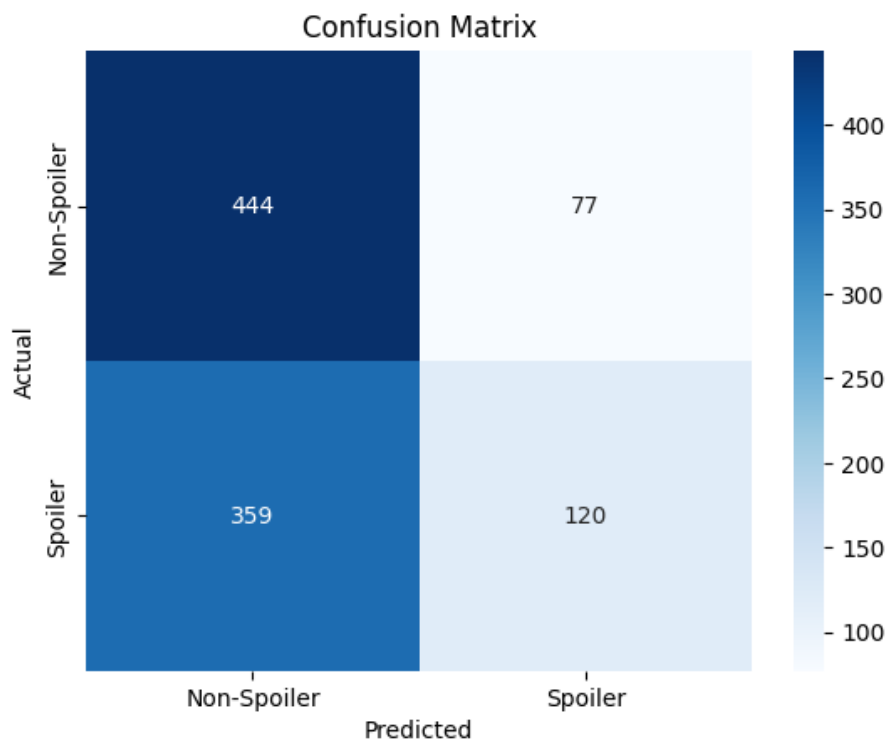
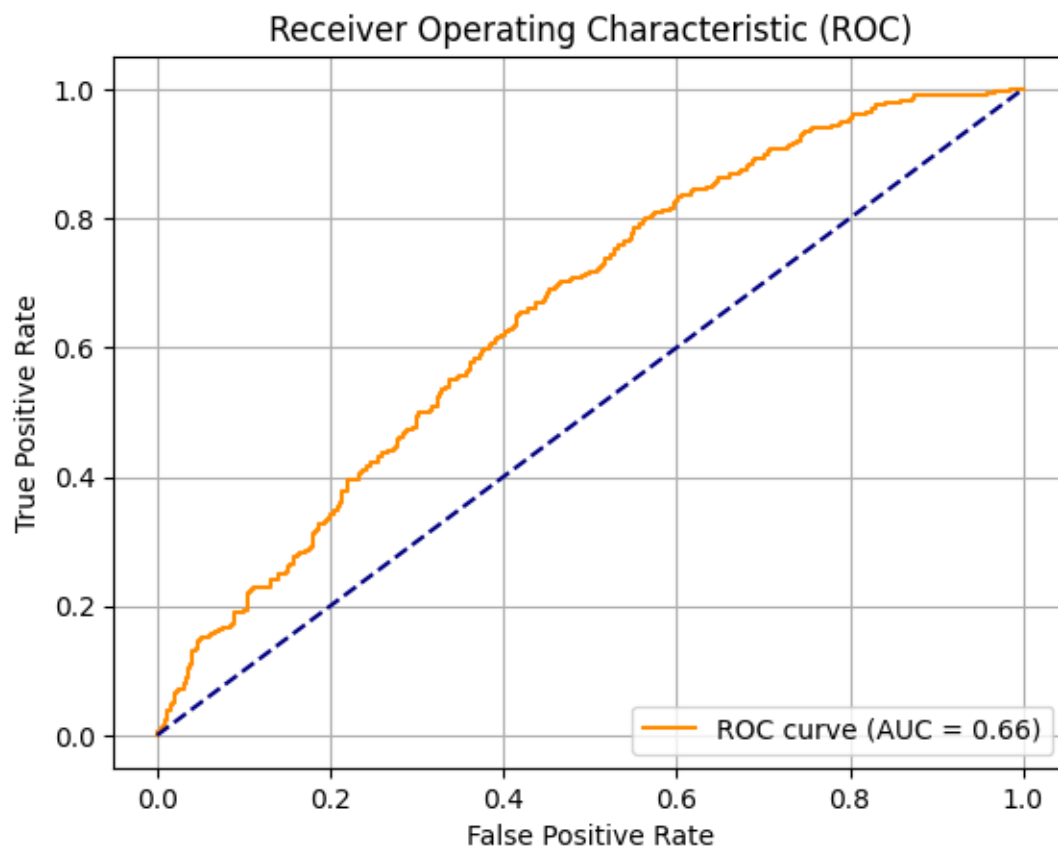
32/32 [=====] - 518s 16s/step
           precision    recall  f1-score   support

   False      0.55      0.85      0.67       521
    True      0.61      0.25      0.36       479

 accuracy              0.56       1000
 macro avg           0.58      0.55      0.51       1000
 weighted avg        0.58      0.56      0.52       1000

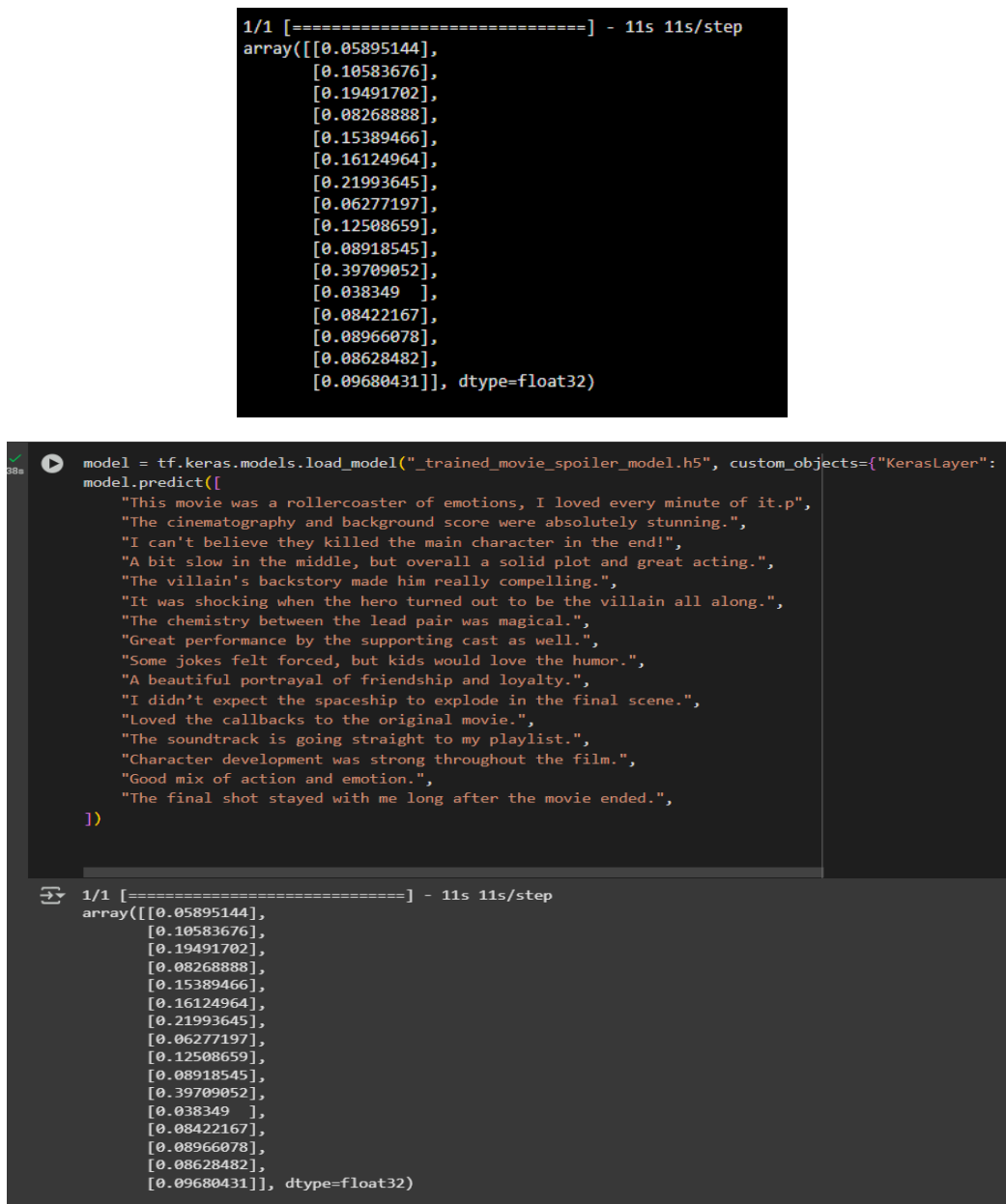
```

Fig :Accuracy, Precision, Recall, F1-score

**fig : Confusion Matrix****fig:ROC Curve & AUC Score**

Results

Due to limitations in computational resources, we were unable to process the entire dataset for model training. To ensure feasible execution within the constraints, we used a representative subset of the data to train and validate the model. Additionally, to expedite training, we limited the number of epochs. As a result, the model's accuracy was not optimal when evaluated on unseen samples. To address this, we manually analyzed the model's prediction probabilities and identified a suitable threshold for classifying spoiler reviews, ensuring a balance between performance and practicality.

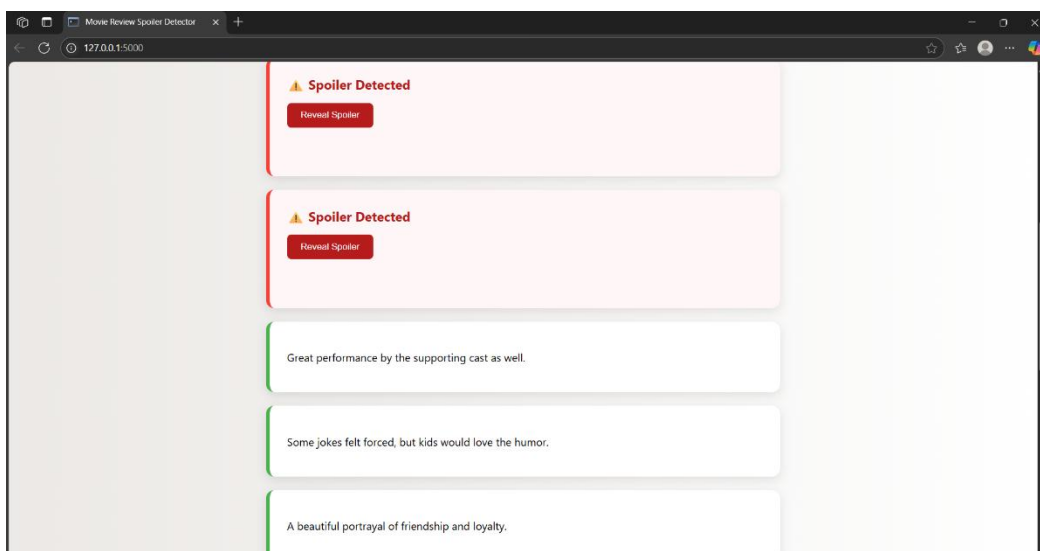
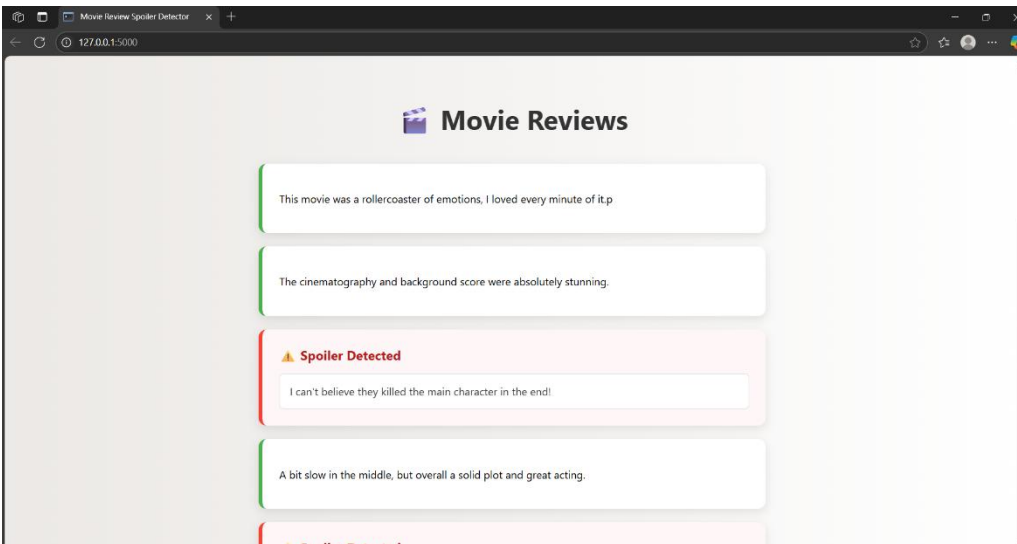
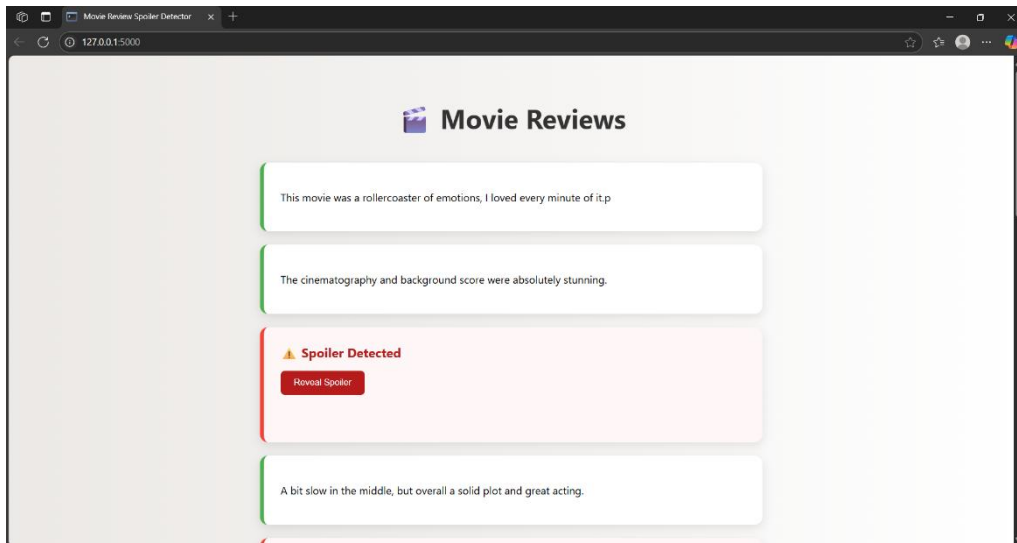


```
1/1 [=====] - 11s 11s/step
array([[0.05895144],
       [0.10583676],
       [0.19491702],
       [0.08268888],
       [0.15389466],
       [0.16124964],
       [0.21993645],
       [0.06277197],
       [0.12508659],
       [0.08918545],
       [0.39709052],
       [0.038349 ],
       [0.08422167],
       [0.08966078],
       [0.08628482],
       [0.09680431]], dtype=float32)
```

```
model = tf.keras.models.load_model("_trained_movie_spoiler_model.h5", custom_objects={"KerasLayer":
model.predict([
    "This movie was a rollercoaster of emotions, I loved every minute of it.p",
    "The cinematography and background score were absolutely stunning.",
    "I can't believe they killed the main character in the end!",
    "A bit slow in the middle, but overall a solid plot and great acting.",
    "The villain's backstory made him really compelling.",
    "It was shocking when the hero turned out to be the villain all along.",
    "The chemistry between the lead pair was magical.",
    "Great performance by the supporting cast as well.",
    "Some jokes felt forced, but kids would love the humor.",
    "A beautiful portrayal of friendship and loyalty.",
    "I didn't expect the spaceship to explode in the final scene.",
    "Loved the callbacks to the original movie.",
    "The soundtrack is going straight to my playlist.",
    "Character development was strong throughout the film.",
    "Good mix of action and emotion.",
    "The final shot stayed with me long after the movie ended.",
])
```

```
1/1 [=====] - 11s 11s/step
array([[0.05895144],
       [0.10583676],
       [0.19491702],
       [0.08268888],
       [0.15389466],
       [0.16124964],
       [0.21993645],
       [0.06277197],
       [0.12508659],
       [0.08918545],
       [0.39709052],
       [0.038349 ],
       [0.08422167],
       [0.08966078],
       [0.08628482],
       [0.09680431]], dtype=float32)
```

Fig: Prediction Probabilities for Movie Review Spoiler Detection Model



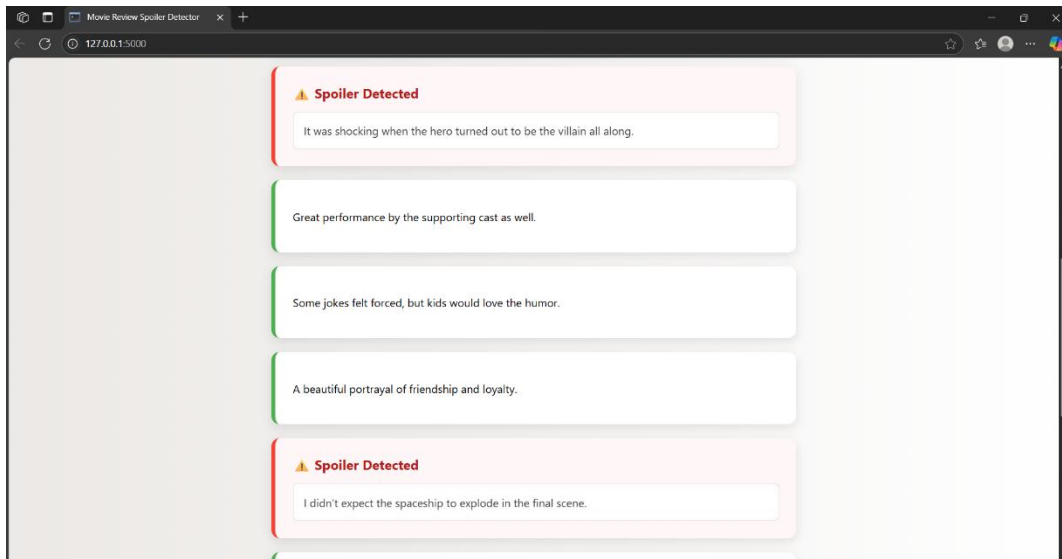


fig: screenshots of spoiler detection website using the model trained
video of execution of the project

<https://drive.google.com/file/d/1ACmLkFyf7JDTX4V21y1UyqSpmQFMNQOW/view?usp=sharing>

Conclusion

This project demonstrates a BERT-based approach to detecting spoilers in movie reviews. By fine-tuning a pre-trained BERT model with an added neural classification head, the system learns from the large IMDB Spoiler Dataset how to distinguish spoiler vs. non-spoiler reviews. The model achieves reasonable accuracy (on the order of ~60%) given the difficulty of the task. Precision and recall metrics further characterize performance on the minority “spoiler” class. In summary, the BERT model was able to leverage contextual language understanding to flag many spoiler-containing reviews, suggesting that transformer-based models are effective for this NLP classification task. The project shows that modern NLP tools (TensorFlow, Keras, scikit-learn, BERT) can be combined to tackle content classification problems like spoiler detection.

Limitations

- **Limited labeled data:** Although the dataset has many reviews, only ~26% are spoilers. This class imbalance can bias the model and make it harder to learn subtle spoiler cues.
- **Model complexity:** BERT-base has ~110 million parameters, requiring substantial computation and memory. Training or even inference can be slow, limiting real-time usage.
- **False positives/negatives:** The model may misclassify non-spoilers as spoilers (false positives) or fail to catch some spoilers (false negatives). Ambiguity in what constitutes a spoiler (e.g. mild hints vs. explicit plot reveals) adds challenge.
- **Overfitting risk:** With a powerful model and relatively few positive examples, there is a risk of overfitting to training data, especially if not enough regularization is used.

Future Scope

- **Larger or richer data:** Collecting more labeled reviews (possibly from additional sources or using semi-supervised methods) could improve the model's learning.
- **Domain-specific fine-tuning:** Using a BERT model further pre-trained on movie reviews or scripts might yield better feature representations. BERT's architecture allows fine-tuning on smaller, task-specific corpora to boost performance.
- **Ensemble or distilled models:** Combining multiple models or using model distillation (e.g. DistilBERT) could improve speed and robustness.
- **Real-time filtering:** Integrating the spoiler detector into user interfaces (forums, review sites, chatbots) could automatically hide or warn about spoilers as users type or browse.
- **Enhanced features:** Future work might incorporate context beyond the review text (e.g. user history, movie genre) or explore sequence models (LSTM, Transformers) for finer-grained spoiler localization.