NLP Course Project – Final Report

Team members:
Prashantan Darshan Naidoo 223009965
Tiara Devanathan 223025166
Mahir Syed 223018507

Topic: Multi-label movie genre classification

Introduction:

The classification of movies into genres is a fundamental task in the film industry for the development of recommender systems, search filters and audience targeting. However, most movies do not fit into a single genre- they span across multiple categories, e.g. Horror-Thriller, Comedy-Drama. Accurately identifying overlapping genres requires an understanding of the themes conveyed in the movie's description. This makes the task well-suited for Natural Language Processing, which can analyse text beyond surface-level keywords.

In this project we explore the task of multi-label classification, where each movie may belong to multiple genres. Using an IMDB dataset, we aim to classify movies based on their descriptions, by training models to predict the set of genres that best describe each film.

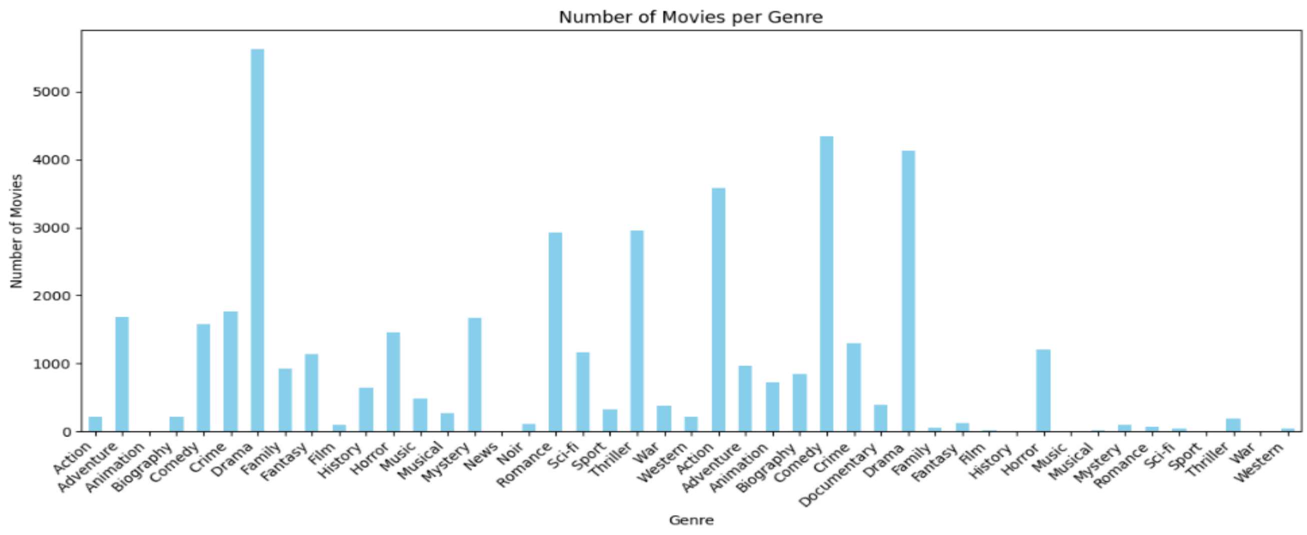
Exploratory Data Analysis:

We used an IMDB dataset containing 18000+ movie genres and their descriptions along with their name. This dataset was obtained from Kaggle.

In this phase, we focused on gaining insights into the distribution of genres in our dataset and understanding the textual content of the movie descriptions.

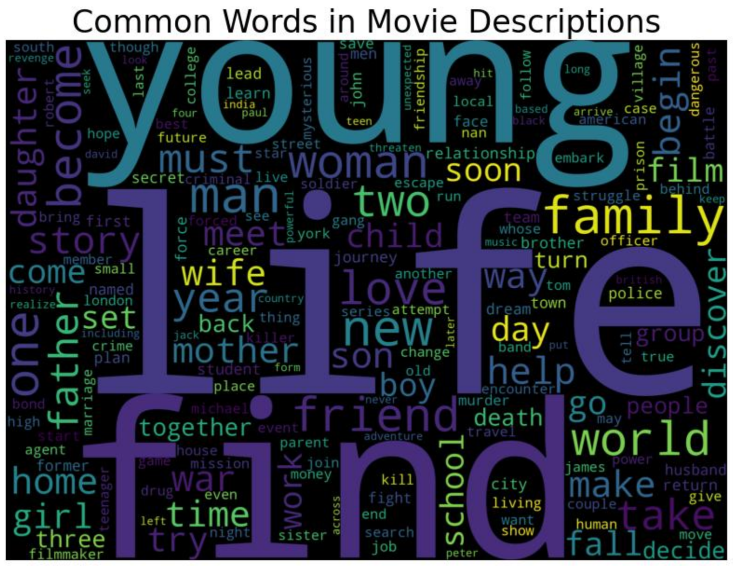
1. Genre distribution

The plot of number of movies per genre revealed an imbalanced distribution, where genres like 'Drama' dominate, while others such as 'Western' have significantly fewer samples. This imbalance is important to address during model training, as it can bias the classifiers toward the more frequent genres. Techniques such as class weighting may be required to mitigate this issue.



1. <u>WordCloud of movie descriptions</u>

To explore the common terms used across all movie descriptions, we generated a word cloud using the WordCloud library. This visualization highlights the most frequent words in the corpus, helping us identify not only standard stop words but also domain-specific terms that may not be informative (e.g. generic words like ‘find’, ‘life’). Insights from this step guided our preprocessing choices, particularly when tuning the parameters of our TF-IDF vectorizer (e.g. setting min_df and max_df to filter out overly common or rare terms).



<u>Data Pre-processing</u>

<u>Data Importing and Filtering</u>

The *importText()* method loads the dataset and filters out entries where the genre is null. From the filtered data we retain only the *genres* and *descriptions*.

<u>Data Cleaning</u>

The following functionalities were performed using the *cleanText()* method:

- converting the text to lowercase
- tokenizing it into individual words
- filtering out non-alphabetic tokens
- removing common english stop words using nltk resources.
- normalized the vocabulary, using the WordNetLemmatizer, reducing the words to just their root word.

Genre Data Conversion

Since the possible genres are stored as comma-separated strings, we transformed this target variable into a binary multi-label format, suitable for our models. We achieved this using our *genreToBinary()* function, which created a matrix where each column represented a unique genre, and each movie's row contained 1s for its associated genres and 0s otherwise. This enables compatibility with various multi-label machine learning models.

Dealing with frequently appearing words

To convert the text into numerical features, we employed the TF-IDF method using scikit-learn's TfidfVectorizer. This would ensure that words are scored based on how important they are to the document they are found in as well as the entire corpus.

The following parameters were used:
- max_features = 5000 to limit to the most informative 5000 features.
- ngram_range (1,2) to capture both unigrams and bigrams.
- min_df = 10 to remove rare words appearing in less than 10 documents.
- max_df = 0.7 to remove overly common words appearing in more than 70% of documents.

This resulted in our primary feature matrix.

Choice of NLP/ML techniques:

We considered 3 NLP/ML techniques, each with different strengths and trade-offs:

1. Logistic Regression

Logistic Regression with the One-vs-Rest strategy, is a scalable choice for classification. It adapts easily to multi-label tasks by training one independent classifier per label. We selected this model as a baseline, to allow us to establish a performance reference point, before moving onto models that handle label dependencies better.

Pros:	Cons:
• Scales well for multi-label problems • Fast to train	• Assumes labels are independent, which is unrealistic for co-occurring genres • Limited to modelling linear relationships between features and labels

2. Classifier Chain with Random Forest

To account for label dependencies, we explored Classifier Chains with Random Forest Classifier as the base learner. Unlike, OvR, Classifier Chains predict labels sequentially, allowing each classifier to consider previous label predictions. We selected this model because label dependency is an important characteristic in movie genres (e.g. "Sci-Fi" often pairs with "Adventure"). Classifier Chains provided a principled way to model such structure, improving on the limitations of Logistic Regression.

Pros:	Cons:
• Uses the input features and the predictions of previous classifiers • Flexible to different base classifiers	• More computationally expensive than OvR, due to chaining • Greedy in nature – errors made early in the chain can propagate, affecting later predictions.

We chose Random Forest Classifier as the base classifier because:

- It performs well on high-dimensional sparse data like TF-IDF.
- It is robust to overfitting.

3. Multi-Layer Perceptron

To further improve modelling capacity, we considered a non-linear, deep learning approach. We chose to use a Multi-Layer Perceptron (MLP) which is a simple deep neural network, trained on scaled TF-IDF features. MLP offered power and simplicity. It can learn complex feature-label and label-label interactions, without relying on sequential chaining. Given the overlapping of genres, the non-linear modelling capacity of MLP made it an appropriate model in our study.

Pros:	Cons:
• Avoids label error propagation problems that can occur in classifier chains	• Requires tuning parameters, such as the learning rate • Slower to train • Requires scaled TF-IDF features

The parameters were set as follows:

- 2 Hidden layers (128 neurons in the first layer, 64 neurons in the second layer)
- ReLU activation function
- Trains with Adam (Adaptive Movement Estimation)
- Trains up to 50 epochs
- Early stopping to prevent over fitting

APIs:

These APIs streamlined the implementation process by providing robust tools for data handling, text preprocessing, visualization, and machine learning, eliminating the need to build complex functionality from scratch.

pandas: Used to load, manipulate and filter dataset.

nltk: For removing common English stop words, splitting descriptions into individual words and reduce words into their base forms.

seaborn: creates a heatmap to visualize per-genre accuracy for model predictions.

matplotlib.pyplot : Plots word clouds, number of movies per genre (bar chart), hamming loss bar chart, and accuracy heatmaps.

numpy: Calculates per-genre accuracy and supports data transformations for plotting and evaluation.

wordcloud: generates word clouds from movie descriptions.
STOPWORDS - remove common but uninformative words.

Scikit-learn(sklearn):
model_selection - split the datasets into training and testing sets
feature_extraction - convert text data into numerical features using TF-IDF
preprocesing - scales features for models like MLP that are sensitive to feature magnitude.
linear_model – logistic regression for classification.
multioutput – allows single-label classifiers to be used for multi-label classifiers.
ensemble – enables the use of the Random Forest Classifier for classification.
preprocessing – scale the data required for MLP training and testing.
neural_network – enables the use of Multi-Layer perceptron for classification.
metrics – imports metrics to evaluate the performance of models.

Empirical evaluation of the NLP system:

We decided to use the following metrics.

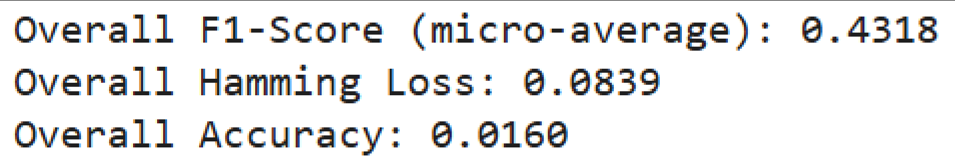
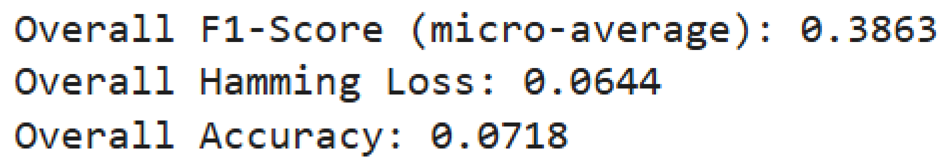
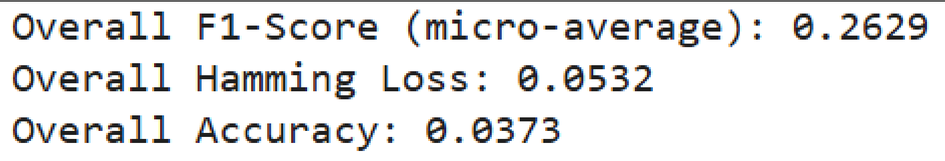
F1-Score: Balances precision (how many predicted genres are correct) and recall (how many of the true genres are recovered). This harmonic mean is critical in multi-label classification, where false positives (predicting extra genres) and false negatives (missing valid genres) both impact the user experience.

Hamming loss: Calculates the fraction of labels that are incorrectly predicted (either missing a label or predicting an extra one) averaged over all labels. This metric is well-suited for multi-label classification, because it is sensitive to both false positives and false negatives, aligning with our need to minimize incorrect genre assignments.

Accuracy: Measures the proportion of samples where all predicted genres exactly match the true genre set. This is a strict metric, as one incorrect or missing genre causes the prediction to be counted as wrong. To complement this, we also generated heatmaps showing per-genre accuracy for each model, providing a more detailed view of performance across individual genres.

Final Results and Conclusion:

The following results were obtained:

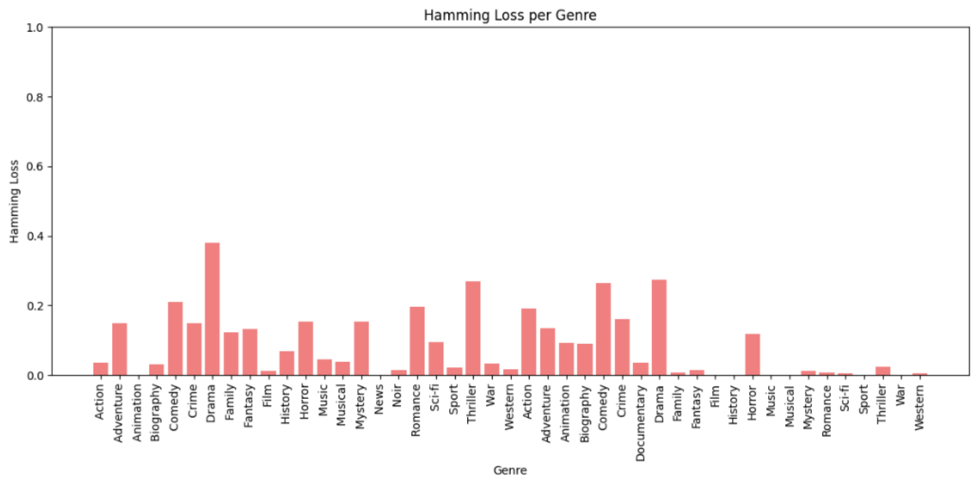
Logistic Regression	 Overall F1-Score (micro-average): 0.4318 Overall Hamming Loss: 0.0839 Overall Accuracy: 0.0160
Classifier Chains	 Overall F1-Score (micro-average): 0.3863 Overall Hamming Loss: 0.0644 Overall Accuracy: 0.0718
Multi-layer Perceptron	 Overall F1-Score (micro-average): 0.2629 Overall Hamming Loss: 0.0532 Overall Accuracy: 0.0373

Comparing these results, we get:

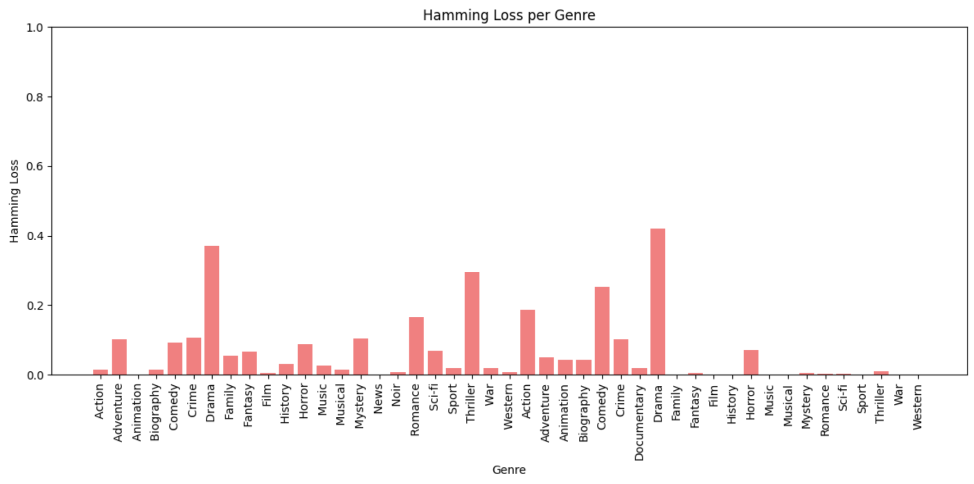
Model	F1-Score	Hamming Loss	Accuracy
Logistic Regression	High	High	Low
Classifier Chains	Medium	Medium	High
Multi-Layer perceptron	Low	Low	Medium

Hamming Loss visualization:

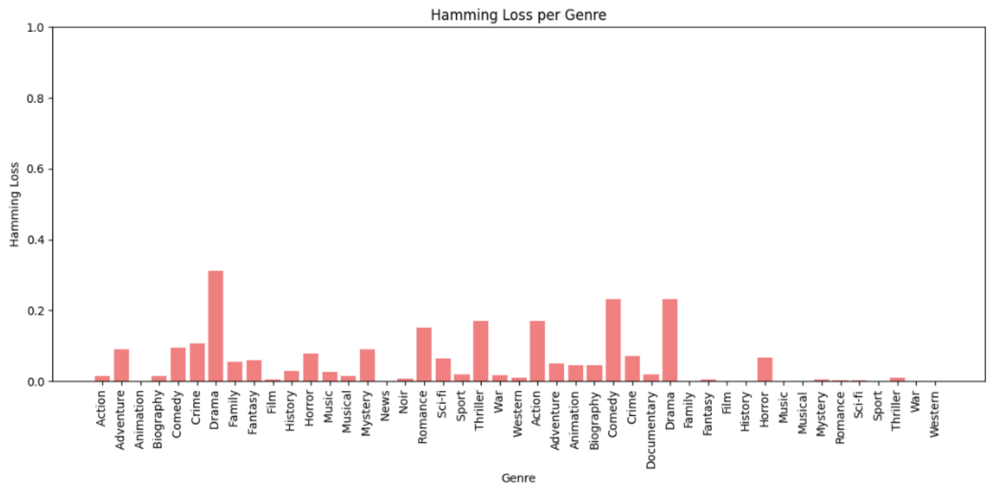
Logistic Regression:



Classifier Chains:

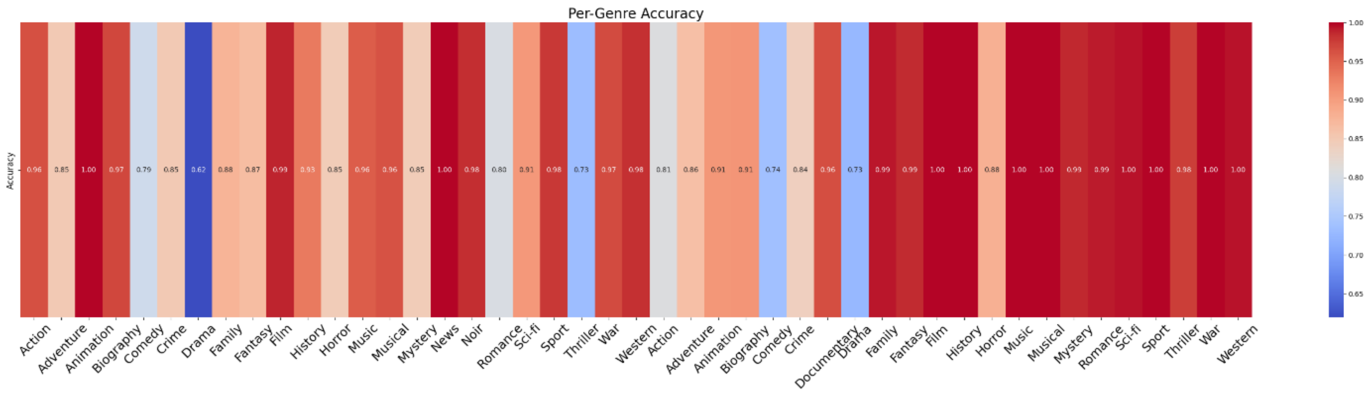


Multi-layer Perceptron:

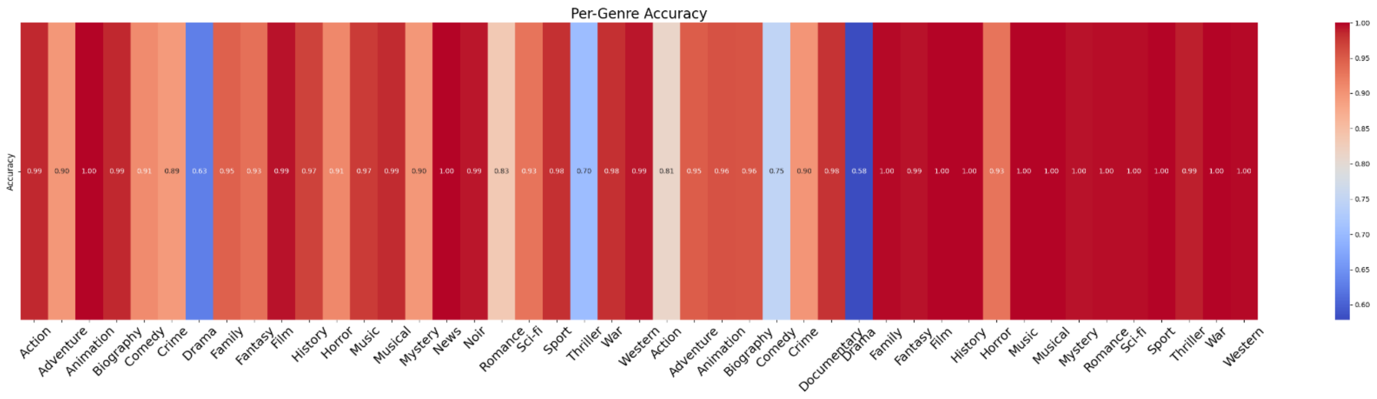


<u>Per-genre accuracy heatmaps:</u>

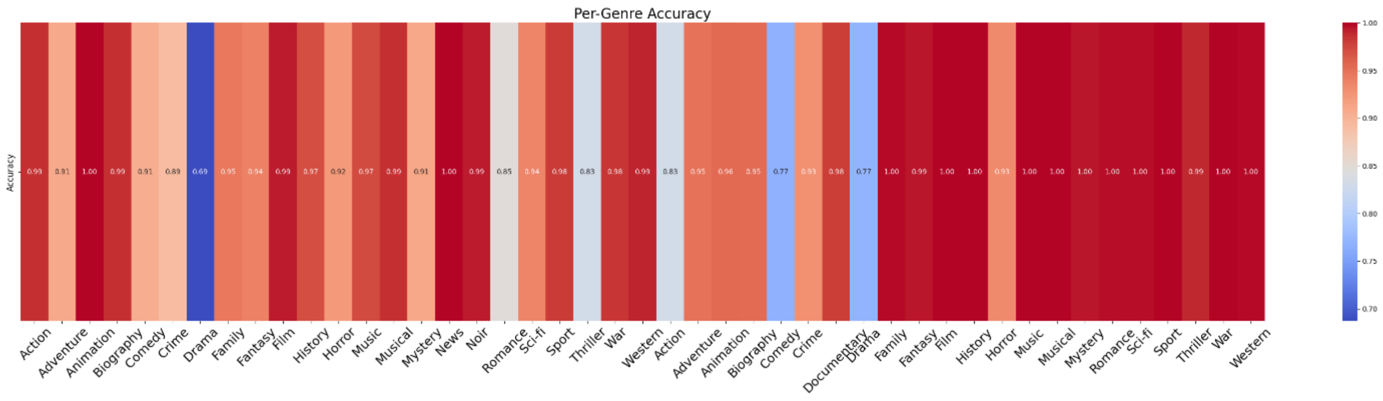
Logistic Regression:



Classifier Chains:



Multi-layer Perceptron:



Observations:

-Logistic Regression had the highest F1-Score, and the highest Hamming Loss, and the lowest overall Accuracy. The accuracy per genre was also low. This indicates that Logistic Regression is acting as a recall-heavy classifier – predicting more genres than are needed.

-Classifier Chains had a medium F1-Score, medium Hamming Loss and highest overall Accuracy. The accuracy per genre was higher than that of Logistic Regression. This suggests that Classifier Chains acts as a precision heavy, conservative classifier- predicting fewer labels but matching more samples exactly.

- Multi-Layer Perceptron had the lowest F1-Score, lowest Hamming Loss, and medium overall Accuracy. The accuracy per genre is higher than that of Classifier Chains. Low F1 score suggests that the model is struggling with balancing precision and recall. Low Hamming Loss means it's making fewer wrong labels overall — possibly by underpredicting genres. Medium Accuracy and better per-genre accuracy suggest it's doing decently at individual labels but failing at matching full label sets.

Conclusion:

For the task of multi-label movie genre classification, **Classifier Chains** emerged as the best-performing model. It effectively balances precision and recall, avoiding both over-prediction and under-prediction, and achieves the highest exact match accuracy. Although we initially expected the deep learning approach (Multi-Layer Perceptron) to outperform Classifier Chains, our results show that this was not the case in this setting.

This suggests that traditional methods like Classifier Chains can remain highly competitive for multi-label classification, especially on smaller datasets where deep learning models may require further tuning and larger training sizes to reach their full potential.