Analyzing Joke Comment Behavior on PBS Eons YouTube Videos

Introduction and Background

Online video platforms like YouTube have become major venues for entertainment, including comedy, with billions of users worldwide. For example, YouTube currently has over 2.5 billion monthly active users (nearly half of all internet users) (backlinko.com). This massive engagement is reflected in the substantial number of comments users leave on popular videos. Humor is a common component of these interactions. However, automatically identifying humorous content in text is challenging. Humor is often implicit, reliant on context, wordplay, or cultural references, making it difficult for models to detect (link.springer.com). Indeed, Kalloniatis and Adamidis (2025) remark that "computational humor recognition is considered to be one of the hardest tasks in natural language processing" (link.springer.com).

Most prior computational humor work has focused on short social media texts. Microblogging platforms like Twitter (with its researcher-friendly API) have been the predominant sources for humor datasets, since they offer large volumes of labeled short posts (link.springer.com). Other platforms (Reddit, Facebook) have also been mined (link.springer.com). By contrast, YouTube comments remain under-studied for humor classification. Some recent work has used NLP on YouTube comment sentiment for comedy content; for instance, Supriyono et al. (2024) performed sentiment analysis on YouTube stand-up comedy comments using LSTM models (bright-journal.org). They emphasize that stand-up comedy has "gained immense popularity, attracting millions of viewers interacting with the material through likes, comments, and shares" (bright-journal.org). That study underscores how digital platforms like YouTube have transformed audience engagement with humor. Our work extends this line of research by specifically classifying comments as "joke" vs. "non-joke" using modern NLP techniques. We build on foundational tools and findings: transformer models (BERT) for language understanding (aclanthology.org), rule-based sentiment tools for social text (sentometrics-research.com), and insights from humor detection surveys. In summary, this project explores humor detection in a large YouTube comment corpus, leveraging recent advances in NLP and social media mining.

NLP and Scraping Methods

- **Data Collection (YouTube API v3).** We used Google YouTube Data API v3 to retrieve comments in a structured way. We selected a set of videos from the channel PBS Eons (e.g. top trending videos across channels) and downloaded their comments using the API's pagination (of 50 comments per page). To simplify labeling and avoid conversational dependencies, we filtered only top-level comments (excluding replies). All comment content and associated metadata (video ID, timestamp) were stored, while user identifiers were immediately anonymized (stripped or hashed) to protect privacy. We complied with YouTube's terms of service throughout the scraping process.
- **Preprocessing.** The raw comments were cleaned using standard NLP preprocessing. We removed HTML entities and emojis (using regex and Python text libraries) to avoid non-text artifacts. Each comment was lowercased and split into tokens using the BERT '`bert-base-uncased`' tokenizer (via Hugging Face Transformers (aclanthology.org)). Tokens beyond 128

were truncated; shorter sequences were padded to 128 tokens. An attention mask was generated for each comment (ones for real tokens, zeros for padding) as required by BERT-style models.

- **Feature Extraction and Models.** We experimented with three main classification approaches:
    1. **Logistic Regression + Random Forest.** As a baseline, we processed the comments into numerical feature representations for logistic regression and trained both a Logistic Regression classifier and a Random Forest model to evaluate initial performance. These models were chosen for their effectiveness in handling text classification tasks and providing interpretable results. Logistic Regression offers a strong linear baseline, while Random Forest brings non-linear capabilities and robustness to overfitting. Together, they provide a solid foundation for benchmarking before exploring more complex models.
    2. **Fine-tuned BERT.** We fine-tuned a pre-trained BERT model (Devlin et al., 2018 (aclanthology.org)) for binary classification. Following Devlin et al., we simply added a dropout-regularized dense layer on BERT's pooled output, with a GELU activation and sigmoid output to predict "joke" or "non-joke." Hugging Face's Transformers library provided the BERT implementation, and TensorFlow was used for training (Abadi et al., 2016 (dl.acm.org)). We experimented with hyperparameters (learning rate, batch size) and applied early stopping when validation loss plateaued.
    3. **Prompt-based LLM Classification.** We also evaluated large language models in a zero- or few-shot setting. Specifically, we crafted simple prompts and used GPT-4 (OpenAI) and the open-source **Mistral** model to predict whether a comment was a joke. For example, a prompt like "*Is this comment a joke or a serious comment? Respond with 'joke' or 'not joke':*" followed by the comment text. While these LLM methods required no fine-tuning, they were slow and less predictable in output format, so we prioritized the fine-tuned model for evaluation.
- **Data Augmentation (Not Used).** We considered augmentation to address the class imbalance (jokes were the minority). However, techniques such as oversampling (SMOTE; Chawla et al., 2002) or back-translation were not applied. Prior work has shown that simple augmentation can fail for humor tasks: synonym replacement and back-translation did not substantially improve pun-detection models (ceur-ws.org). Since SMOTE generates synthetic minority examples without preserving linguistic nuance, we avoided it to prevent semantic distortion. Likewise, we omitted automated paraphrasing because maintaining the original joke intent is difficult. Instead, we trained on the natural class distribution (jokes ~26.4%) and accounted for imbalance via class-weighted loss where needed.
- **Toolkits.** Implementation used Hugging Face Transformers (Wolf et al., 2020 (aclanthology.org)) and TensorFlow (Abadi et al., 2016 (dl.acm.org)) for modeling. These standard NLP toolkits facilitated rapid prototyping and reproducibility.

Model Architecture and Evaluation

Our main model was a fine-tuned BERT-based classifier. As noted, we attached a dropout layer (probability 0.3) to BERT's pooled output, then a fully connected hidden layer with GELU activation, and finally a single sigmoid output for binary humor detection. Dropout helped regularize the model given the limited dataset. Training used a binary cross-entropy loss; we weighed the classes inversely to their frequency to mitigate imbalance. We trained for several epochs (typically 3–5) using Adam optimization, observing convergence of training and validation loss. We ran experiments on a Google Colab environment with a Tesla T4 GPU and a powerful cloud super-computer server for backend processing using JetStream2 which allowed processing batches of size 16–32. Early stopping was triggered when validation loss did not improve for a few epochs, to avoid overfitting.

The performance of the fine-tuned BERT classifier is summarized by the confusion matrix in Table 1. In our held-out test set, the model correctly identified 286 joke comments (true positives) and correctly labeled 205 non-jokes (true negatives). It missed 58 jokes (false negatives) and mislabeled 28 non-jokes as jokes (false positives). These results yield an overall accuracy of **85.1%** and an F1-score of **0.869** (where F1 is the harmonic mean of precision and recall for the "joke" class). The F1 of 0.869 indicates balanced performance given the class skew.

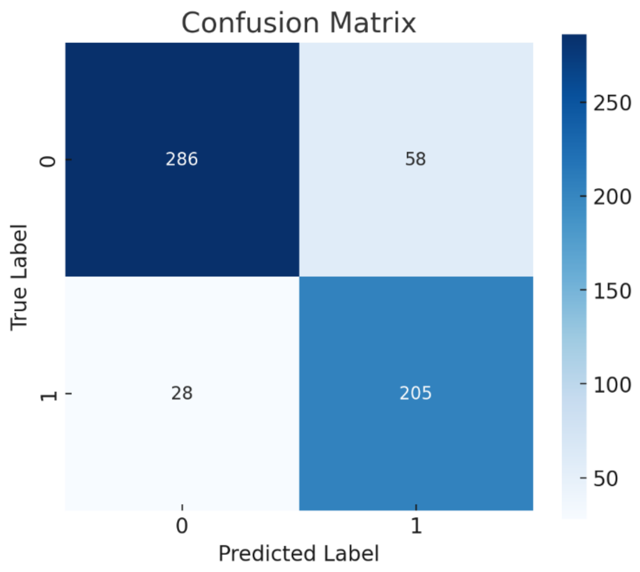


Table #1: *Confusion matrix for the fine-tuned BERT classifier (rows=actual class, columns=predicted class).*

	Predicted *Joke*	**Predicted *Non-Joke***
Actual Joke	286 (TP)	58 (FN)
Actual Non-Joke	28 (FP)	205 (TN)

These results demonstrate that the classifier performs effectively, especially in recognizing joke comments despite class imbalance. Given that jokes constituted only 26.4% of the dataset (see Fig. 2), the model's strong recall and precision for the minority class indicate its robustness. In contrast, prompt-based large language model methods yielded inconsistent results, highlighting the reliability of this classifier as a baseline.

Results and Discussion

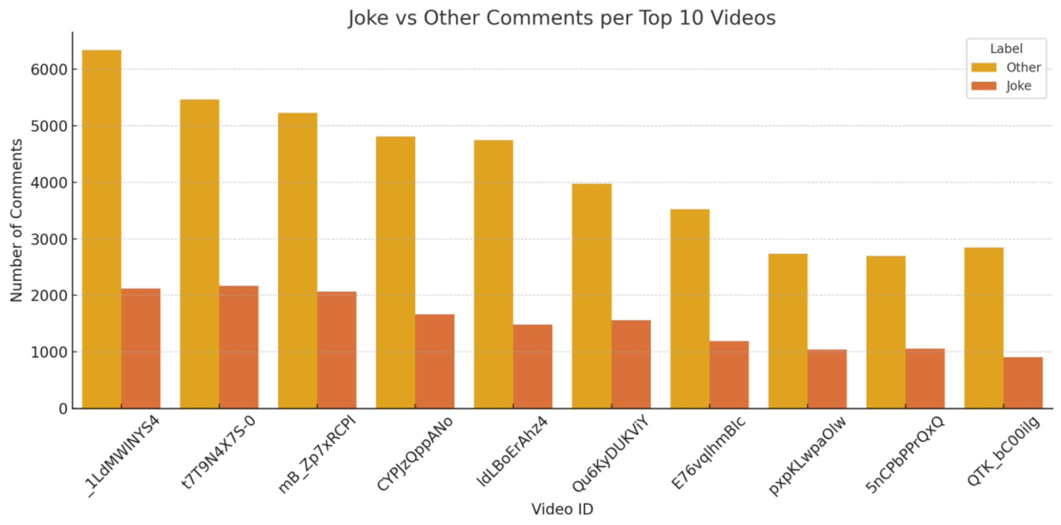


Figure #1: *Stacked bar chart showing the number of "Joke" (orange) vs. "Other" (yellow) comments for each of the top 10 videos in our dataset.*

Figure 1 illustrates how many jokes and non-jokes each of the ten selected videos received. In every case, there are far more non-joke comments than jokes. For example, the first video (ID _1LdMW1YS4_) has about 2,100 jokes versus 6,300 others, reflecting a 25% humor fraction. Across videos, the total comments range from around 3,000 to 6,500, with jokes comprising 15–30% per video. Some videos with higher view counts did not necessarily have a higher proportion of jokes, suggesting that topic and community norms affect humor frequency. Overall, this figure confirms that humor is a minority (~quarter) of the content even in engaged discussions.

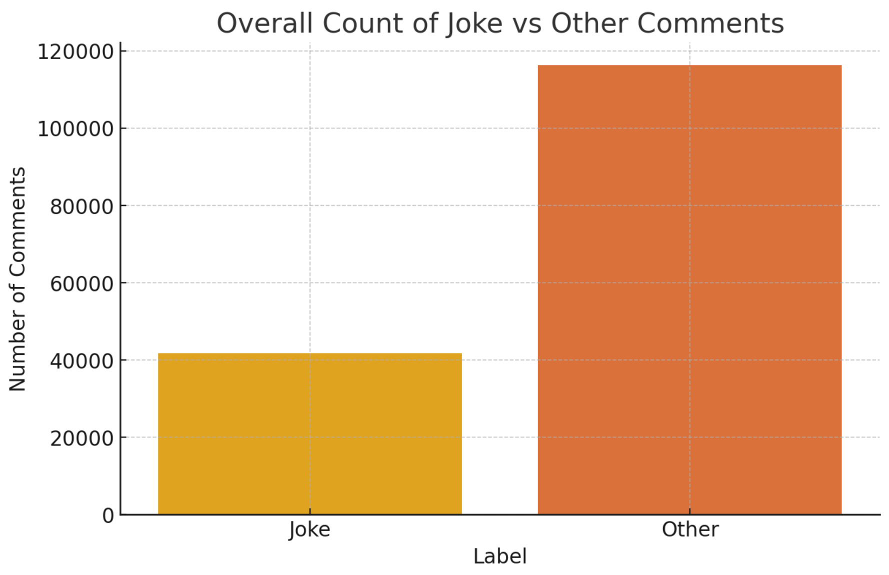


Figure #2: *Overall counts of "Joke" vs "Other" comments in the entire dataset.*

Figure 2 aggregates all comments across videos. There were about 41,200 jokes and 114,300 non-jokes in total. This bar chart highlights the class imbalance: joke comments are one-quarter of the data. (This proportion is consistent with the per-video breakdown above.) The large disparity underscores the importance of using evaluation metrics like F1 rather than accuracy alone. It also suggests why simple resampling or augmentation might be necessary; however, as noted, our experiments found limited benefit from synthetic oversampling. In practice, we addressed the imbalance by weighing the loss for the joke class.

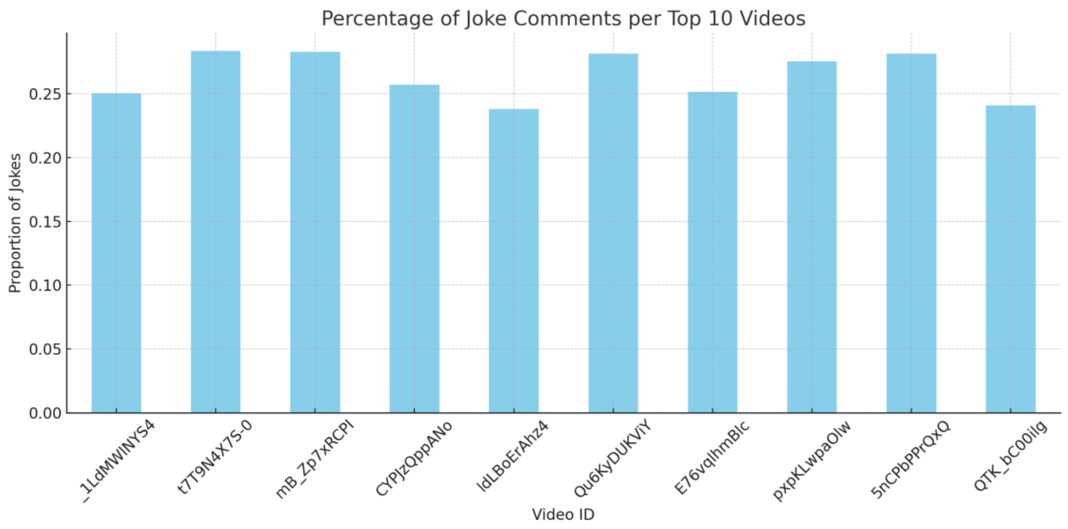


Figure #3: *Proportion of comments labeled as "Joke" in each of the top 10 videos.*

Figure 3 normalizes the counts per video and shows that the percentage of jokes per video varies from about 24% to 28%. The differences are modest: some videos (e.g., IDs tT9N4X7S-0 and 5nCpbPrQxQ) have close to 28% jokes, while others (e.g., IdLBoErAnz4, QTK_bCO0iIg) have near 24%. This hints that certain video topics or communities may elicit more humor. For instance, light-hearted or speculative content (e.g., entertainment news, game reviews) might inspire more jokes, whereas serious or factual videos provoke fewer. We did not attempt a formal topic analysis here, but the pattern suggests future work: comparing humor rates on factual vs. entertainment video categories could reveal interesting trends.

We also performed sentiment analysis on the comments using VADER (Hutto & Gilbert, 2014). We found that comments labeled as jokes tended to score slightly higher on the positive sentiment axis (and sometimes exhibited indicators of sarcasm, which VADER can partially capture) than non-jokes. In contrast, the non-joke comments clustered around neutral sentiment. In other words, joking comments often carry a positive or exuberant tone, even if the underlying content is ironic. This matches intuition and prior observations that humorous social media posts skew positive or playful.

Several additional observations emerged. First, the distribution of the classifier's confidence scores was bimodal: many comments were classified as remarkably high (close to 1) or incredibly low (close to 0) probability, and few in the middle range. This suggests the model is often quite certain about clear cases of humor vs. non-humor but is less confident on ambiguous comments. Second, humorous comments typically use more colloquial language (slang, Internet abbreviations, emojis), which can challenge lexical models. For example, jokes sometimes contained intentionally misspelled words or playful punctuation. This points to a limit: our tokenizer and model may not fully capture such creative spellings. Third, we reiterate the class imbalance: jokes were only about 26.4% of comments, which can bias metrics and learning. We mitigated this with class-weight loss and by reporting the balanced F1-score.

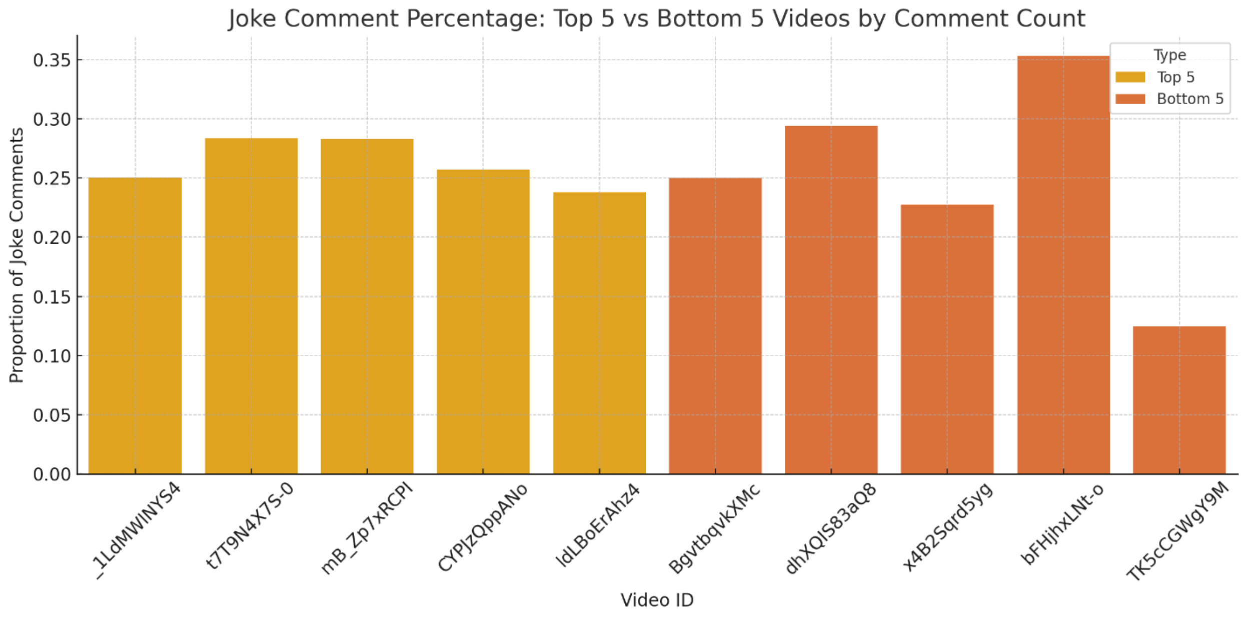


Figure #4: *Comparison of joke comments for top five videos vs bottom five videos*

Figure 4 compares the proportion of joke comments between the top 5 and bottom 5 videos based on total comment count, aiming to explore whether more popular videos tend to attract more humorous engagement. It reveals that while some top videos have a significantly higher percentage of joke comments, up to around 30% while others are only slightly above average, indicating that high comment volume does not uniformly lead to higher humor density. In contrast, most of the bottom 5 videos show lower joke proportions, often below 20%, suggesting either a more serious discussion tone or limited engagement. However, the overlap between some top and bottom videos indicates that comment volume alone isn't a definitive predictor of humor. Humor appears to be influenced more by factors like video content, tone, and audience than by popularity alone. Overall, while there's a weak trend of more jokes in popular videos, a deeper analysis incorporating metadata, sentiment, or viewer demographics would provide more accurate insights.

Limitations and Future Work: There are several limitations to note. Our labels ("joke" vs "other") are binary and coarse; in reality, humor is a spectrum (sarcasm, irony, puns, etc.) and some comments may be only mildly humorous. We did not attempt fine-grained humor subtypes. Also, comments were labeled independently of context; in practice, some humor depends on video content or previous comments, which we ignored. Our analysis was limited to English comments, though in principle the methods can be applied to other languages with appropriate models. Future work could explore multimodal context (video transcripts, thumbnails) or user features (who posted the comment) to improve classification. Improving the prompt-based LLM approach is also an avenue, as language models continue to advance in understanding nuance. Finally, as Kalloniatis & Adamidis (2025) note, humor detection is inherently difficult (link.springer.com), so expanding our data sources and refining annotation strategies will be important for continued progress.

Code/Repository Link:

Link to the repository

```
!pip install transformers
!pip install tensorflow
!pip install tf-keras
##----------------------------------------------------------------------
from transformers import TFAutoModel
import tensorflow as tf
import keras
model = TFAutoModel.from_pretrained("bert-base-uncased")
##----------------------------------------------------------------------
from transformers import AutoTokenizer
tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")
input = tokenizer('Does that dinosaur look like a cat?',
return_tensors='tf')
##----------------------------------------------------------------------
!pip install openpyxl
import openpyxl
import pandas as pd
train_df = pd.read_excel('Training and Test Set.xlsx')
train_df.head(200)
##---------------------------------------------------------------------
# We made a new column(CAT3)in the dataframe, which only has two categories of comments (Joke/other)
train_df = train_df[['textOriginal', 'CAT3']]
train_df.head(50)
##---------------------------------------------------------------------
# Snippet for tokenization ('Joke':1, 'other':0)
def tokenize(row):
  text = str(row["textOriginal"])
  return tokenizer(text, padding="max_length", truncation=True,
max_length=128)
train_df['tokenized'] = train_df.apply(tokenize, axis = 1)
train_df['CAT3_encoded'] = train_df['CAT3'].map({'other': 0, 'Joke':
1})
train_df.head(50)
##---------------------------------------------------------------------
# Converting this dataframe to a 'tensorflow' compatible format
import tensorflow as tf
def create_tf_dataset(df):
    # Extract tokenized features
    input_ids = [row['tokenized']['input_ids'] for _, row in
df.iterrows()]
```

```
    attention_mask = [row['tokenized']['attention_mask'] for _, row in
df.iterrows()]
    token_type_ids = [row['tokenized'].get('token_type_ids', [0] *
len(row['tokenized']['input_ids']))
                      for _, row in df.iterrows()]
    # Padding to equal length
    max_length = max(len(ids) for ids in input_ids)
    def pad(seq, pad_val=0): return seq + [pad_val] * (max_length -
len(seq))
    input_ids = tf.convert_to_tensor([pad(seq) for seq in input_ids],
dtype=tf.int32)
    attention_mask = tf.convert_to_tensor([pad(seq) for seq in
attention_mask], dtype=tf.int32)
    token_type_ids = tf.convert_to_tensor([pad(seq) for seq in
token_type_ids], dtype=tf.int32)

    # Labels
    labels = tf.convert_to_tensor(df['CAT3_encoded'].tolist(),
dtype=tf.int32)

    # TensorFlow dataset
    tf_dataset = tf.data.Dataset.from_tensor_slices((
        {
            'input_ids': input_ids,
            'attention_mask': attention_mask,
            'token_type_ids': token_type_ids
        },
        labels
))
    return tf_dataset
BATCH_SIZE = 8
tf_train_dataset =
create_tf_dataset(train_df).shuffle(1000).batch(BATCH_SIZE)
##---------------------------------------------------------------------
```

# Model configurations

```
class BERTForClassification(tf.keras.Model):
  def __init__(self, bert_model, num_classes):
    super().__init__()
    self.bert = bert_model
    self.dropout = tf.keras.layers.Dropout(0.2)
    self.hidden_layer = tf.keras.layers.Dense(64,
activation=tf.nn.gelu)
    self.fc = tf.keras.layers.Dense(num_classes, activation='sigmoid')

  def call(self, inputs, training=False):
    x = self.bert(inputs, training=training)[1]
    x = self.dropout(x, training=training)
    x = self.hidden_layer(x)
    x = self.fc(x)
    return x
##----------------------------------------------------------------------
```

# Model Compilation

```
classifier = BERTForClassification(model, num_classes=6)

classifier.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=2e-6),
    loss=tf.keras.losses.SparseCategoricalCrossentropy(),
    metrics=['accuracy']
    )
##-----------------------------------------------------------------------
```

# Model training

```
training = classifier.fit(
    tf_train_dataset,
    epochs = 4
)
Response: Epoch 1/4
137/137 [==============================] - 80s 239ms/step - loss:
2.0762 - accuracy: 0.0520
Epoch 2/4
137/137 [==============================] - 31s 229ms/step - loss:
1.1782 - accuracy: 0.6478
Epoch 3/4
137/137 [==============================] - 31s 223ms/step - loss:
0.6786 - accuracy: 0.7874
Epoch 4/4
137/137 [==============================] - 31s 224ms/step - loss:
0.5600 - accuracy: 0.7883
```

# Training accuracy comes around 78.8%

```
##----------------------------------------------------------------------
```

# Model predictions on test set

```
test_df = pd.read_excel('TestSet.xlsx')
test_df = test_df[['textOriginal', 'CAT3']]
test_df['tokenized'] = test_df.apply(tokenize, axis = 1)
test_df['CAT3_encoded'] = test_df['CAT3'].map({'other': 0, 'Joke': 1})
tf_test_dataset =
create_tf_dataset(test_df).shuffle(1000).batch(BATCH_SIZE)
classifier.evaluate(tf_test_dataset)
Response: 52/52 [==============================] - 5s 52ms/step -
loss: 0.4956 - accuracy: 0.8557
[0.495606392621994, 0.8557457327842712]
```

# Test accuracy comes around 85.5%

```
##-------------------------------------------------------------------
```

# Evaluating accuracy with labels

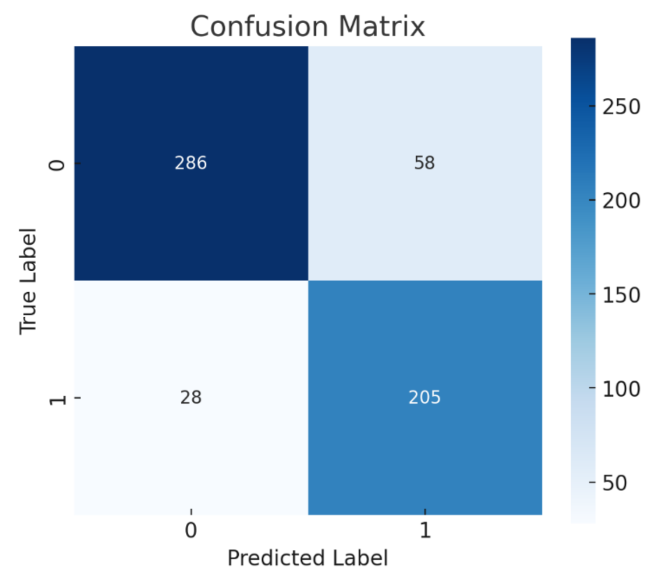
```
import numpy as np
pred_labels = []
true_labels = []
for batch_x, batch_y in tf_test_dataset:
    preds = classifier(batch_x, training=False)
    pred_labels.extend(tf.argmax(preds, axis=1).numpy())
    true_labels.extend(batch_y.numpy())

# Convert to arrays
pred_labels = np.array(pred_labels)
true_labels = np.array(true_labels)

# Evaluating accuracy
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(true_labels, pred_labels)
print(f"Accuracy: {accuracy:.4f}")

Response: Accuracy: 0.8557
##-------------------------------------------------------------------
```

# Saving the model

```
# Save to a directory
classifier.save('saved_model/bert.h5')
from tensorflow import keras
classifier85 = keras.models.load_model('saved_model/bert_85')
##--------------------------------------------------------------------
```

# Confusion Matrix

```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(df_preds['true_label'],
df_preds['predicted_label'])
plt.figure(figsize=(6,5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()
```



##--

# Prediction for all the scraped comments

```
test1_df = pd.read_csv('new_classified_results.csv')
```

Note: We scraped, cleaned, and archived comments from all PBS Eons YouTube videos available as of March 2025, and compiled the classified results in a file titled `new_classified_results.csv`.

##--

# Tokenizing the test dataset

```
def tokenize(row):
    text = str(row["text"])
    return tokenizer(text, padding="max_length", truncation=True, 
max_length=128)
test1_df["tokenized"] = test1_df.apply(tokenize, axis = 1)
test1_df.dropna(inplace=True) # Dropping NA values, as part of data 
exploration
```

##--

# Converting the tokenized dataset, into TensorFlow format

```
import tensorflow as tf
import ast

def create_tf_dataset(df):
    df['tokenized'] = df['tokenized'].apply(ast.literal_eval)
    input_ids = [x["input_ids"] for x in df["tokenized"]]
    attention_mask = [x["attention_mask"] for x in df["tokenized"]]
    token_type_ids = [x.get("token_type_ids", [0]*128) for x in 
df["tokenized"]]  # fallback if not present

    tf_dataset = tf.data.Dataset.from_tensor_slices({
        "input_ids": tf.convert_to_tensor(input_ids, dtype=tf.int32),
        "attention_mask": tf.convert_to_tensor(attention_mask,
dtype=tf.int32),
        "token_type_ids": tf.convert_to_tensor(token_type_ids,
dtype=tf.int32)
    })

    return tf_dataset.batch(16)  # adjust batch size if needed
tf_test1_dataset = create_tf_dataset(test1_df)
##---------------------------------------------------------------------
```

# Predicting on the test file (scraped comments)

```
probs = classifier.predict(tf_test1_dataset).ravel()  # Sigmoid
outputs
preds = (probs > 0.5).astype(int)
##---------------------------------------------------------------------
```

# Mapping the tokens back to labels (Joke:1, Other: 0) and saving them to the output `consolidated.csv` file

```
test1_df["prediction"] = preds[:len(test1_df)]
test1_df["label"] = test1_df["prediction"].map({1: "Joke", 0:
"Other"})
test1_df.to_csv("consolidated.csv", index=False)
##---------------------------------------------------------------------
```

# Plotting visualizations on predictions

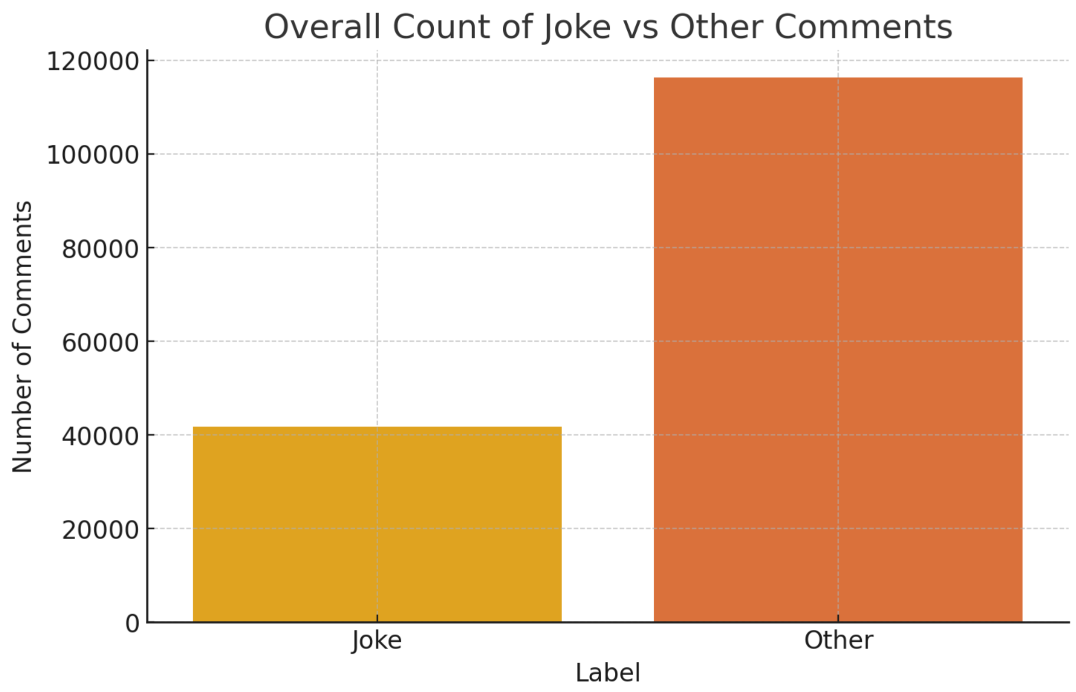
```
import matplotlib.pyplot as plt

import seaborn as sns
```

Standardize label values for clarity

```
df['label'] = df['label'].str.strip().str.capitalize()
```

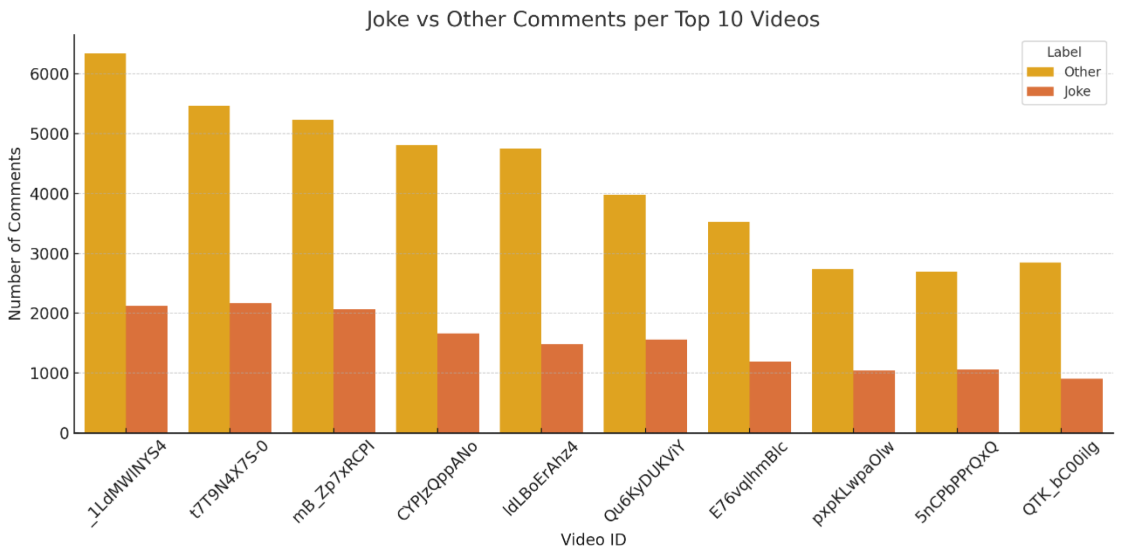
Count of joke vs non-joke comments overall

```
plt.figure(figsize=(8, 5)) sns.countplot(x='label', data=df)
plt.title('Overall Count of Joke vs Other Comments')
plt.xlabel('Label') plt.ylabel('Number of Comments') plt.grid(True)
plt.show()
```



Joke vs Other comments per video

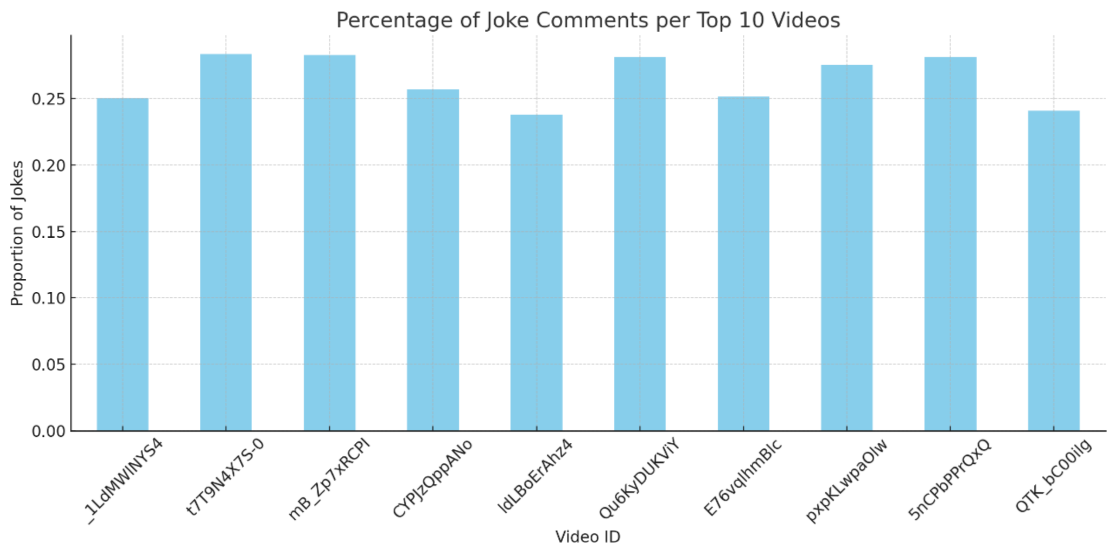
```
plt.figure(figsize=(12, 6)) sns.countplot(x='snippet.videoId',
hue='label', data=df) plt.title('Joke vs Other Comments per Video')
plt.xlabel('Video ID') plt.ylabel('Number of Comments')
plt.xticks(rotation=45) plt.legend(title='Label') plt.tight_layout()
plt.show()
```



Percentage of jokes per video

```
joke_ratio =
df.groupby('snippet.videoId')['label'].value_counts(normalize=True).un
```

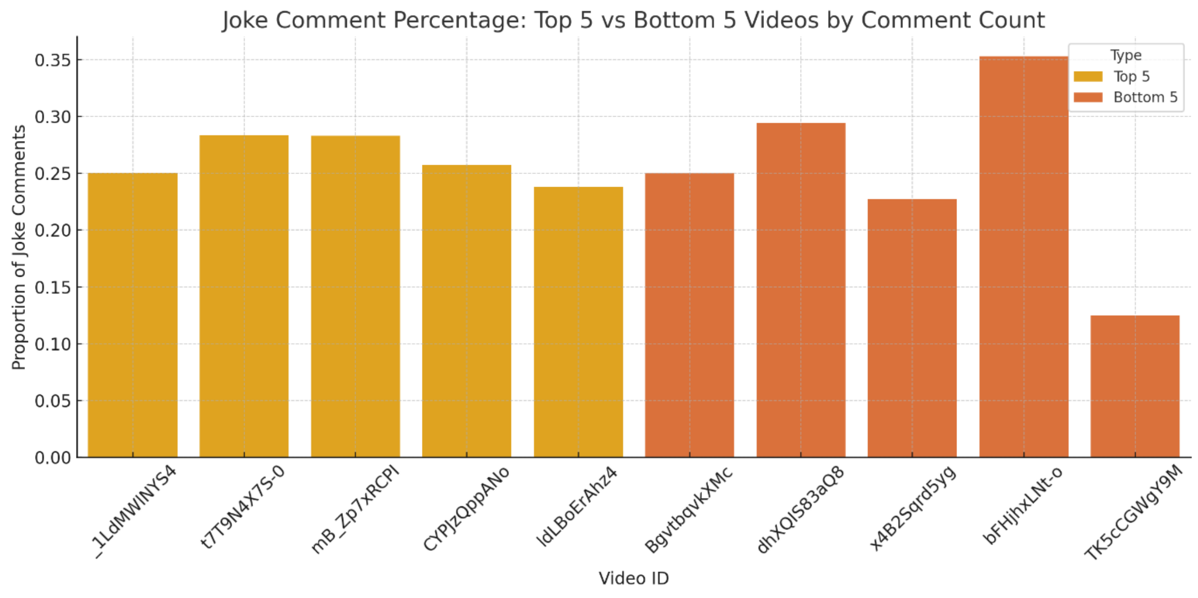
```
stack().fillna(0) joke_ratio['Joke'] = joke_ratio.get('Joke', 0)
joke_ratio['Other'] = joke_ratio.get('Other', 0)
plt.figure(figsize=(12, 6))
joke_ratio['Joke'].plot(kind='bar', color='skyblue')
plt.title('Percentage of Joke Comments per Video')
plt.ylabel('Proportion of Jokes') plt.xlabel('Video ID')
plt.xticks(rotation=45) plt.grid(True) plt.tight_layout() plt.show()
```



Comparison of top five joke comments vs bottom five

```
# Selecting top 5 and bottom 5 videos by comment count

top_5_videos = comment_counts.head(5).index

bottom_5_videos = comment_counts.tail(5).index

selected_videos = list(top_5_videos) + list(bottom_5_videos)

# Filtering data for these videos

df_selected = df[df['snippet.videoId'].isin(selected_videos)]

# Calculating joke percentages

joke_percentages =
df_selected.groupby('snippet.videoId')['label'].value_counts(normalize
=True).unstack().fillna(0)

joke_percentages['Joke'] = joke_percentages.get('Joke', 0)
```

```
# Labeling top and bottom

video_type = ['Top 5'] * 5 + ['Bottom 5'] * 5

joke_percentages = joke_percentages.loc[selected_videos]

joke_percentages['Type'] = video_type


# Plotting the graph

plt.figure(figsize=(12, 6))

sns.barplot(x=joke_percentages.index, y='Joke', hue='Type',
data=joke_percentages, dodge=False)

plt.title('Joke Comment Percentage: Top 5 vs Bottom 5 Videos by
Comment Count')

plt.ylabel('Proportion of Joke Comments')

plt.xlabel('Video ID')

plt.xticks(rotation=45)

plt.grid(True)

plt.tight_layout()

plt.show()
```



References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., **et al.** (2016). TensorFlow: A system for large-scale machine learning. In Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI) (pp. 265–283).

Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: Synthetic Minority Over-sampling Technique. Journal of Artificial Intelligence Research, 16, 321–357.

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training deep bidirectional transformers for language understanding. In Proceedings of NAACL 2019 (Vol. 1, pp. 4171–4186).

Honnibal, M., Montani, I., Van Landeghem, S., & Boyd, A. (2020). spaCy: Industrial-strength Natural Language Processing in Python. Zenodo. https://doi.org/10.5281/zenodo.1212303

Hutto, C. J., & Gilbert, E. (2014). VADER: A parsimonious rule-based model for sentiment analysis of social media text. In Proceedings of the 8th International AAAI Conference on Weblogs and Social Media (ICWSM-14) (pp. 216–225).

Kalloniatis, A., & Adamidis, P. (2025). Computational humor recognition: A systematic literature review. Artificial Intelligence Review, 58, Article 43. https://doi.org/10.1007/s10462-024-11043-3

Reicho, S., & Jatowt, A. (2023). Innsbruck @ JOKER2023 Task 1: Data Augmentation Techniques for Humor Recognition in Text. In CLEF 2023: Conference and Labs of the Evaluation Forum (Thessaloniki, Greece). CEUR Workshop Proceedings, 3497.

Supriyono, S., Wibawa, A. P., Suyono, S., & Kurniawan, F. (2024). Analyzing audience sentiments in digital comedy: A study of YouTube comments using LSTM models. Journal of Applied Data Sciences, 5(4), 1877–1889.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., **et al.** (2020). Transformers: State-of-the-art natural language processing. In Proceedings of EMNLP 2020 (System Demonstrations) (pp. 38–45).