CSCI 3303 MATHEMATICS FOR COMPUTING III

SEMESTER 1 2024/2025

SECTION 4

GROUP REPORT

Random Joke Generator With Adam Optimizer

CONTRIBUTED BY GROUP C

NAME	MATRIC NO
NUR SYAFIKA BINTI BAHRUDIN	2219998
SYASYA BINTI SYAERILL	2213690
UNGKU QISTINA BINTI UNGKU MOHD FARIS	2215442

SUPERVISED BY:

Dr. AKEEM OLOWOLAYEMO

Introduction

The Jokes Generator Project leverages advancements in natural language processing (NLP) and machine learning to create a tool capable of generating humorous text. With the increasing use of artificial intelligence in creative domains, this project seeks to explore the intersection of technology and entertainment, focusing on humor as a unique and challenging aspect of human communication. By training a character-level language model using Long Short-Term Memory (LSTM) networks, the project aims to understand patterns in jokes and generate content that mimics the humor style of the input dataset.

Humor plays a vital role in human interaction, enhancing communication, reducing stress, and fostering social connections. Despite its importance, humor generation is a complex and underexplored area in artificial intelligence due to its subjective and cultural nature. Creating a jokes generator addresses the challenge of replicating this inherently human skill in a computational model. Additionally, the growing demand for AI-driven content creation across various industries underscores the need for tools that can generate diverse and engaging content, including humor. Whether for entertainment, marketing, or education, a jokes generator can provide significant value by automating creative tasks and offering a unique, lighthearted way to interact with AI.

Primary objectives of this project is to develop an AI-powered jokes generator. It is to build and train a character-level language model using LSTM networks capable of generating coherent and humorous text. Next is to explore humor patterns in text. This can be done by analyzing the structure and linguistic features of jokes to enhance the model's ability to replicate humor effectively. Lastly, it is to improve interaction between humans and AI. By creating an accessible interface for users to generate jokes interactively, fostering engagement with AI-based creativity.

This project is expected to yield several benefits, one of them is enhanced understanding of NLP in humor. This study will contribute to research on humor generation, an area that combines linguistic, psychological, and computational challenges. Next is to make use of the practical application of AI in the entertainment industry. This can lead to the development of AI tools for entertainment industries, such as scriptwriting or content creation. Not only that, having this generator should improve user experience with AI by interacting with a humor-focused AI, users may develop a more positive perception of AI technologies.

Related Work

Gabbard and Miller (2018) examines the principles and application of Stochastic Gradient Descent (SGD) and Adam optimizers in the training of neural networks. This study offers a summary of neural networks, describes backpropagation, and assesses the effectiveness of both optimizers in classification and reinforcement learning activities utilizing MATLAB. The research shows that Adam typically exceeds SGD regarding both speed and precision. Furthermore, they explore reinforcement learning, particularly Deep Q-Learning (DQN), and how it's used to train agents for navigating environments. The study seeks to clarify neural network training and offer a basic comprehension of these optimization methods.

van Houdt et. al. (2020) reviewed the Long Short-Term Memory (LSTM) model, emphasizing its influence on machine learning and neurocomputing. They focus on LSTM's capability to manage the vanishing/exploding gradient issue, enabling it to perform well in tasks such as speech recognition, machine translation, and time series forecasting. The analysis discusses LSTM's structure, training techniques, and different applications, such as text recognition, natural language processing, and computer vision. They additionally explore hybrid models that merge LSTM with different neural networks to improve performance. In general, this study highlights LSTM's adaptability and efficiency in managing sequential data across various fields.

Kigma and Ba (2015) present Adam, a method for first-order gradient-driven optimization of stochastic objective functions. Adam merges the benefits of AdaGrad and RMSProp, rendering it efficient, memory-friendly, and appropriate for large-scale issues with noisy or sparse gradients. It utilizes adaptive calculations of first and second moments of gradients and incorporates bias-correction factors. They present theoretical convergence characteristics, empirical evidence showing the effectiveness of Adam, and address a variant known as AdaMax. Adam demonstrates strong performance and is highly appropriate for various machine learning tasks, such as logistic regression, neural networks, and convolutional neural networks.

Staudemeyer and Morris (2019) offer an in-depth guide on Long Short-Term Memory (LSTM) Recurrent Neural Networks (RNNs). They describe the development of LSTM-RNNs, tackling the vanishing gradient issue present in traditional RNNs. The study discusses the LSTM architecture, which includes memory cells, input and output gates, and the constant error carousel (CEC). It also addresses training techniques such as Backpropagation Through Time (BPTT) and Real-Time Recurrent Learning (RTRL). Furthermore, they examine different LSTM extensions and uses, including speech recognition, handwriting recognition, and machine translation, underscoring LSTM's efficiency in managing long-term dependencies in sequential information.

Kazi et. al. (2021) introduces a transformer-based neural joke generator that utilizes pre-trained language models. The method consists of three phases: fine-tuning a GPT-2 model on a dataset of jokes, training a BERT-based classifier to identify sentences as jokes or otherwise, and additionally training the generator alongside the identifier using Proximal Policy Optimization (PPO). The joke generator was optimized using 173,635 jokes, whereas the identifier was developed with a balanced collection of jokes and non-jokes. The training with PPO enhanced the quality of the jokes produced. The joke identifier attained an accuracy of 95.83%, while human reviewers found 26.7% of the created jokes to be humorous.

Toplyn (2021) presents Witscript 2, a system created to produce conversational humor based on common knowledge instead of puns. Using OpenAI's GPT-3, Witscript 2 employs a five-step algorithm for joke creation: choosing a subject, pinpointing two engaging topic handles, generating associations for these handles, crafting a punchline, and finding a perspective to link the topic to the punch line. Assessments by human evaluators indicated that Witscript 2's jokes were categorized as jokes 46% of the time, in contrast to 70% for jokes created by humans. This suggests that Witscript 2 is capable of generating jokes that are contextually integrated and more complex than those based on wordplay. The system showcases computational creativity and commonsense understanding, indicating that creating humor rooted in common sense might not be an AI-complete challenge. Future enhancements focus on boosting Witscript 2's ability to craft jokes, potentially exceeding the quality of wordplay-centric humor.

Suljic and Pervan (2024) explore the convergence of human and artificial intelligence (AI) in creative writing, focusing on the aspect of humor in texts produced by AI. The research employs Bard, an AI text generator, to produce application letters in three types: standard, comedic, and subtly humorous according to particular prompts. The study utilizes the humor transaction framework, emphasizing the generation, expression, and perception of humor. Results show that although AI can generate amusing and entertaining content, it does not possess the originality and creative depth found in humor created by humans. The research also underscores possible biases in AI-produced content and examines the effects for upcoming creative writing, involving concerns about authorship and copyright. The authors determine that humor created by AI can captivate audiences, yet it brings up issues regarding the future of human creativity and the place of AI in literature.

Tiwari et al. (2024) studies how humor influences software development and affects developer involvement. The research analyzes three open-source initiatives: faker, lolcommits, and volkswagen, emphasizing how humor can improve the software development experience. Faker creates funny test data, lolcommits takes developer selfies with every commit, and Volkswagen amusingly alters test outcomes in continuous integration (CI). In a survey conducted with 125 developers, the authors discovered that humor is generally valued and can enhance the enjoyment of coding, promote community spirit, and boost collaboration. Nonetheless, they highlight the importance of using humor responsibly to prevent harmful effects on code quality and professionalism. The research finds that humor, when employed correctly, can greatly boost developer engagement and creativity.

Lee et al. (2017) present a model for producing humour in a temporal and geographical environment, improving conversational AI by using recurrent neural networks with natural language processing (NLP) and understanding.Their approach combines picture processing and linguistic data to create jokes that are contextualised to user trends, enhancing emotional involvement. Applications highlight the significance of computational humour in emotional AI and include psychiatric counselling and stress management. The study promotes the use of machine learning for context-sensitive humour generating while highlighting the drawbacks of previous methods, such as their high cost and dependence on rule-based systems.

Faulkner et al. (2021) presented a neural joke generating and classification system utilising BERT classifiers and GPT-based language model.They use sophisticated natural language processing (NLP) to analyse the structure of jokes, producing humor-focused, context-sensitive phrases and using an offensiveness classifier to filter out improper information. Their strategy combines web applications for gathering user feedback with GPT-2 and dataset-specific fine-tuning for improved joke coherence and naturalness. Experiments show notable improvements over conventional techniques in terms of accuracy in humour and perplexity classification. The work addresses issues like dataset replication and context-driven interactions while highlighting GPT's potential in free-form humour generating.

Yeh et al. (2018) developed a system for generating original puns using recurrent neural networks (RNNs) with a focus on phonetic-based humor generation.Using long short-term memory (LSTM) cells, their method combined word-based and character-based RNN models to preserve the contextual coherence necessary for joke creation. Additionally, they tried with phonetic edit distances and N-gram models for pun production, but their success in punchline construction was limited. The work focused on character-based RNNs for flexibility in managing out-of-vocabulary concerns, using a dataset of jokes from Reddit and other humour websites. Their findings show promise for deep learning-based humour generating, despite the fact that computing constraints prevented lengthy training and ideal performance. Future suggestions include combining phonetic analysis to increase pun quality and allocating more resources for in-depth training.

Chippada and Saha (2018) introduced a novel controlled Long Short-Term Memory (LSTM) architecture for generating both jokes and quotes by training a unified model with categorical input tags.In contrast to earlier unsupervised humour models, their method use category tags to guide sentence construction, allowing for the on-demand production of sentiment-specific content. The dataset uses pre-trained GloVe embeddings for word representation and incorporates tweets, quotes, and jokes from various sources. The network's capacity for semantic learning is demonstrated by controlled LSTM, which enables the creation of forward and backward jokes. Evaluations reveal that mixed-category training and a creative fusion of humour and inspiration result in less inappropriate jokes. This study focuses on combining mixed data and categorical control to improve text generation's coherence and humour quality.

Methodology

This project utilizes a character-level language model built with Long Short-Term Memory (LSTM) neural networks, optimized with the Adam optimizer to generate jokes. The methodology focuses on data preprocessing, sequence modelling, neural network architecture, optimization, and evaluation. The mathematical foundation of optimization in this context, particularly focusing on linear algebra, the Adam optimizer and its links to discrete dynamic systems. Linear algebra is the mathematical foundation underlying most machine learning and deep learning algorithms.

In this project, linear algebra is used to represent and manipulate data, model parameters, and computational operations. One of them is data representation where text data is converted into numerical forms, such as one-hot encoded vectors or embeddings, which are represented as matrices. Next is neural network operations where it computes in neural networks, such as forward propagation, involving matrix multiplications and additions. Lastly is optimization where the weights of the LSTM network are updated using gradients, which are computed through matrix calculus. The general formulation for data representation and one-hot encoding is as follows. The input data is represented as a matrix X, where each row is a one-hot encoded vector. For a vocabulary size of V and sequence length T, each vector x has:

$$x \in \mathbb{R}^{V}, X \in \mathbb{R}^{T\times V}$$

Linear algebra is used to solve problems involving data representation. Representing high-dimensional data in compact mathematical forms. It also involves transformations. That is by applying linear transformations to data, such as through weight matrices in neural networks. Other than that is optimization. This is achieved by solving problems where parameters must be adjusted to minimize or maximise an objective function, such as loss in machine learning models. Lastly is sequence modelling where it represents temporal relationships in data using matrices and vectors.

LSTM networks involve complex computations at each time step to manage long-term dependencies. The core equations for the LSTM cell are:

Forget Gate:

$$f_t = \sigma (W_f \bullet h_{t-1} + U_f \bullet x_t + b_f$$

Input Gate:

$$i_t = \sigma (W_i \bullet h_{t-1} + U_i \bullet x_t + b_i)$$

Candidate Memory:

$$\tilde{C}_t = tanh\,(W_c \bullet h_{t-1} + U_c \bullet x_t + b_c\ )$$

Memory Update:

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$$

Output Gate:

$$o_t = \sigma(W_o \bullet h_{t-1} + U_o \bullet x_t\ + b_o$$

Hidden State:

$$h_t = o_t \odot\ tanh\,(C_t)$$

The Adam Optimizer is used to minimize the categorical cross-entropy loss. It combines momentum-based methods like RMSprop with adaptive learning rates. The weight update rule for Adam is:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)\, g_t$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2\ )g_t^2$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t},\ \hat{v}_t = \frac{\hat{m}_t^{\ 2}}{1-\beta_2^t}$$

$$\theta_t = \theta_{t-1} - \eta\frac{\hat{m}_t}{\sqrt{\hat{v}_t}+\epsilon}$$

Where m_t and v_t are the moving averages of the gradient and squared gradient, respectively. β_1 and β_2 are exponential decay rates. g_t is the gradient loss with respect to the parameter. η is the learning rate. While ϵ is a small constant for numerical stability.

The loss function for multi-class classification is given by:

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{V} y_{ij} log(\hat{y}_{ij})$$

Where N is the number of samples. V is the vocabulary size. On the other hand, y_{ij} is the true label, and $\hat{y}_{ij}$ is the predicted probability.

Optimization

The Adam optimizer is an optimization algorithm widely used in training machine learning models. It combines the strengths of two popular methods: momentum-based optimization and adaptive learning rates. In essence, Adam updates the model's parameters by minimizing a loss function, which, in this project, is the categorical cross-entropy loss. It calculates two moving averages at each iteration: one for the gradient (direction and magnitude of change needed) and another for the squared gradient (scale of updates). These averages help the optimizer adjust the learning rate for each parameter dynamically. The weight update rule for Adam involves using these adjusted rates to ensure faster convergence and stability, even with sparse gradients or noisy data. Mathematically, it relies on exponential decay rates (β1 and β2) to compute weighted averages of gradients and incorporates a small constant (ϵ) for numerical stability, ensuring parameter tuning.

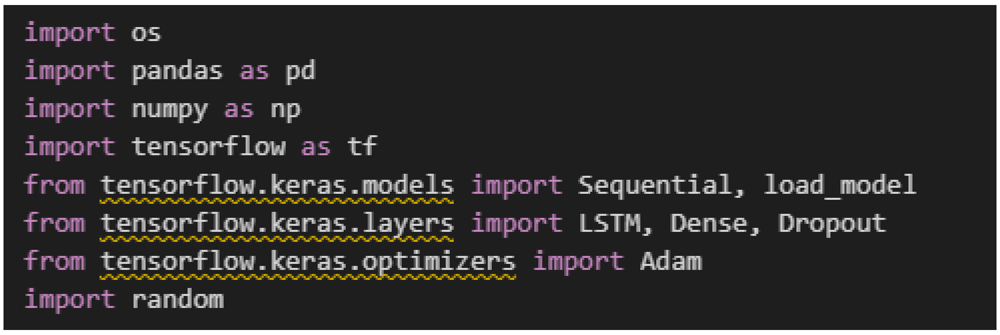
An LSTM network functions as a discrete dynamic system by modelling the evolution of states (hidden states and memory cells) over a series of discrete time steps. In this context, the input sequence drives the state transitions, governed by non-linear transformations such as the forget, input, and output gates. Each state is influenced by its immediate predecessor and the current input, resembling the progression of a dynamic system where the next state depends on the current state and an external stimulus. The equations describing the LSTM's operations serve as discrete mappings that capture temporal dependencies in sequential data, analogous to how discrete dynamic systems describe the behavior of variables over time. This connection enables the LSTM to effectively capture long-term dependencies and generate coherent outputs, making it particularly suited for sequence modeling tasks like text generation.

Variable and Data Source

Data Source

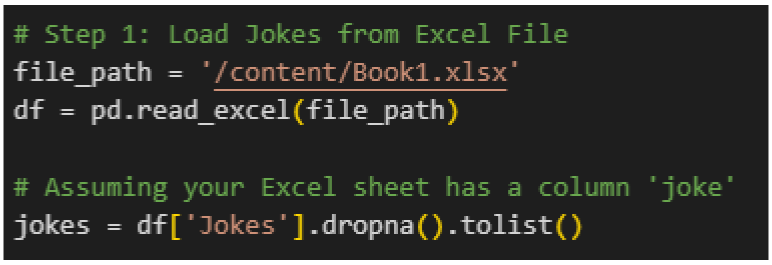
For this project, the dataset that was used to train the joke generator model was self-generated by our team. We collaboratively created a collection of 99 jokes to ensure that the dataset aligned with the specific requirement of the project. We carefully created the jokes to make sure everyone can understand it and recorded the jokes inside an Excel file, which served as our primary data source. Each joke was structured in a simple text format to ensure that the train process will be easier and compatible with the LSTM model. The decision to self-generate the dataset was made to ensure that we got the format of the jokes as we want and make it easy for everyone else to understand the jokes. There are many dataset that we can find in Kaggle.com, but most of them are in the format of questions and answers. So it is difficult for us to use that kind of dataset in our project as it is not a one sentence joke like the self-generated jokes.

Variable



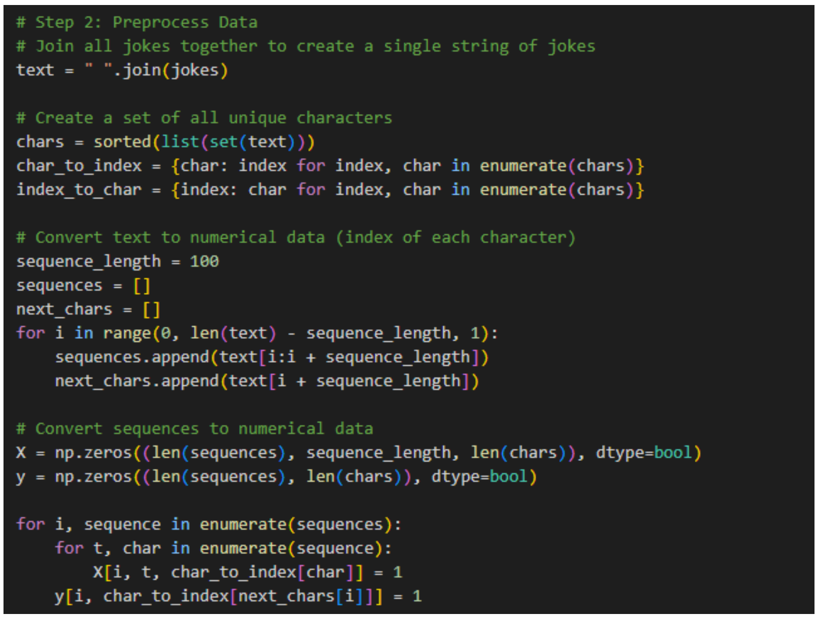
```python
import os
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.layers import LSTM, Dense, Dropout
from tensorflow.keras.optimizers import Adam
import random
```

This part of the code brings in all the tools needed to create and run the joke generator. The **os** module helps manage files and folders, like loading or saving data. Next, **pandas** and **numpy** are used to handle and process data because **pandas** works well with tables of data, and **numpy** is great for math and working with numbers. The **tensorflow** library, together with **Keras** tools are being used to build the neural network. We also includes layers like **LSTM** to understand the sequence, **Dense** to produce outputs, and also **Dropout** to prevent the model from overlifting. Other than that, we also include an **Adam** optimizer for it to help train the model by adjusting how it learns. Lastly, we import **random** modules to pick random items. All the tools are needed to create and train the joke generator.



```python
# Step 1: Load Jokes from Excel File
file_path = '/content/Book1.xlsx'
df = pd.read_excel(file_path)

# Assuming your Excel sheet has a column 'joke'
jokes = df['Jokes'].dropna().tolist()
```

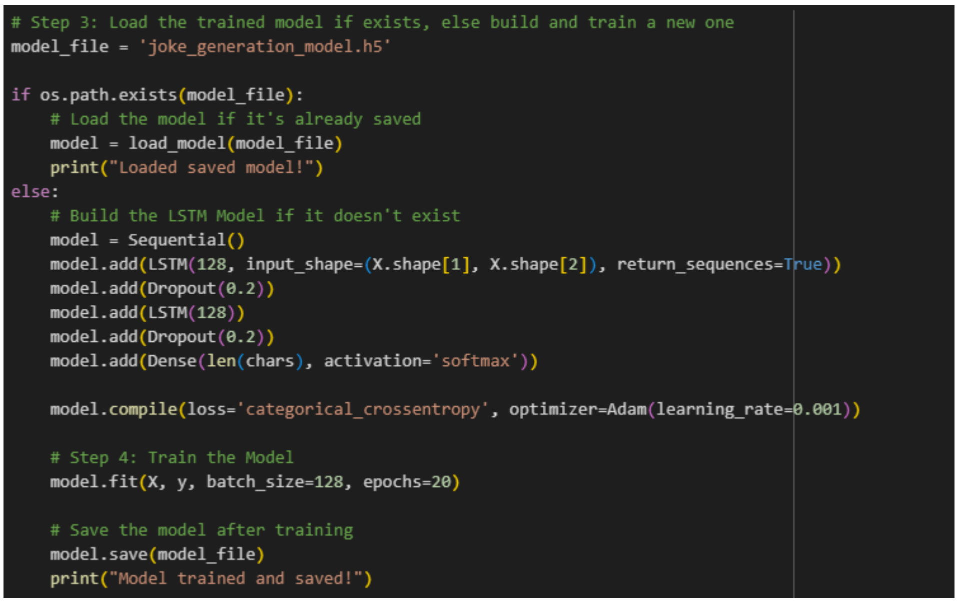
This part of the code is responsible for loading the jokes from an Excel file into the program. First, the **file_path** variable specifies the location of the Excel file **"Book1.xlsx"** that contains the jokes dataset. Then, using the **pd.read_excel(file_path)** function from the **pandas** library, we will use it to read the file and store its data in a DataFrame called **df**. The code assumes that the jokes are stored in a column named "Jokes". To make sure that there will be no missing or empty entries, we used the **.dropna()** function and also the **.tolist()** function that will convert the cleaned column of jokes into a Python list, which is a great format for data processing. At the end of this step, the jokes variable holds all the jokes from the Excel file in a neat list.



```python
# Step 2: Preprocess Data
# Join all jokes together to create a single string of jokes
text = " ".join(jokes)

# Create a set of all unique characters
chars = sorted(list(set(text)))
char_to_index = {char: index for index, char in enumerate(chars)}
index_to_char = {index: char for index, char in enumerate(chars)}

# Convert text to numerical data (index of each character)
sequence_length = 100
sequences = []
next_chars = []
for i in range(0, len(text) - sequence_length, 1):
    sequences.append(text[i:i + sequence_length])
    next_chars.append(text[i + sequence_length])

# Convert sequences to numerical data
X = np.zeros((len(sequences), sequence_length, len(chars)), dtype=bool)
y = np.zeros((len(sequences), len(chars)), dtype=bool)

for i, sequence in enumerate(sequences):
    for t, char in enumerate(sequence):
        X[i, t, char_to_index[char]] = 1
    y[i, char_to_index[next_chars[i]]] = 1
```

This part of the code prepares the jokes for training the LSTM model by converting them into numerical data that the model can understand. First, all the jokes in the list are joined together into a one large string using **" ".join(jokes)**. This will create a continuous sequence of text. Then, it will identify all unique characters in the text and store them in a sorted list called **chars**. Two dictionaries are created here, **char_to_index** and **index_to_char**. Both are used to map the character to its unique number (index) and reverse.

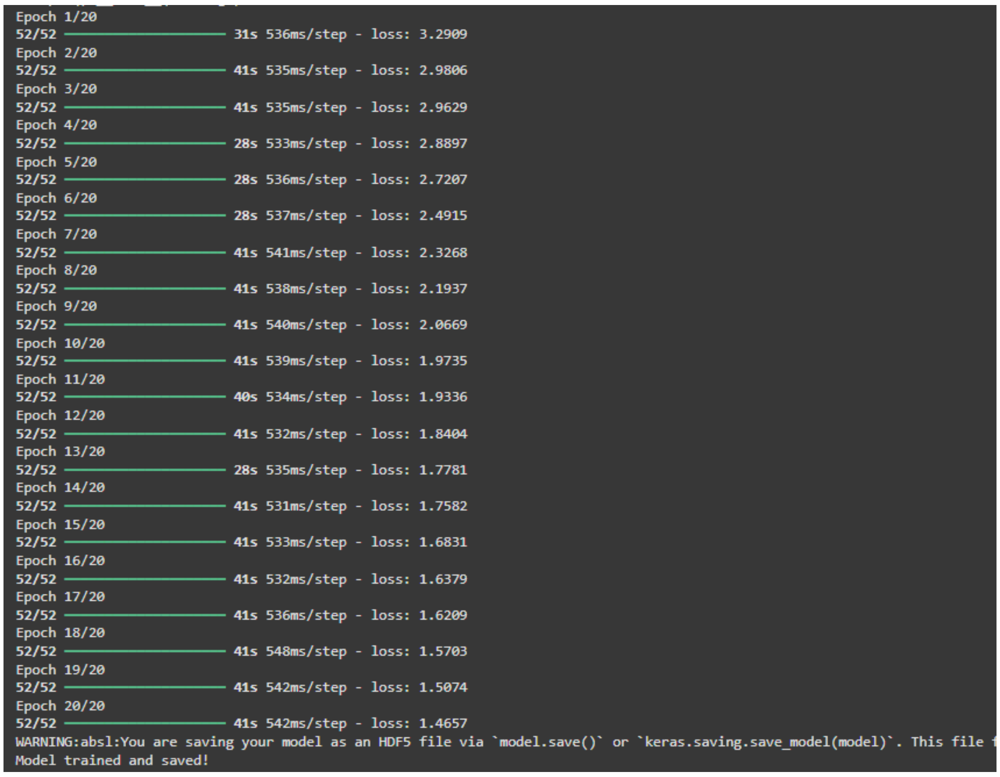
Next, the text is split into smaller chunks called “sequence”. Each sequence will consist of 100 characters long, and the model will use these sequences to learn the pattern in the text. Then we create the sequences list to store the 100 character chunk and the next_chars list to store the character that comes immediately after each sequence. Here is the essential part to prepare the data for training the LSTM model to generate the jokes.



```python
# Step 3: Load the trained model if exists, else build and train a new one
model_file = 'joke_generation_model.h5'

if os.path.exists(model_file):
    # Load the model if it's already saved
    model = load_model(model_file)
    print("Loaded saved model!")
else:
    # Build the LSTM Model if it doesn't exist
    model = Sequential()
    model.add(LSTM(128, input_shape=(X.shape[1], X.shape[2]), return_sequences=True))
    model.add(Dropout(0.2))
    model.add(LSTM(128))
    model.add(Dropout(0.2))
    model.add(Dense(len(chars), activation='softmax'))

    model.compile(loss='categorical_crossentropy', optimizer=Adam(learning_rate=0.001))

    # Step 4: Train the Model
    model.fit(X, y, batch_size=128, epochs=20)

    # Save the model after training
    model.save(model_file)
    print("Model trained and saved!")
```

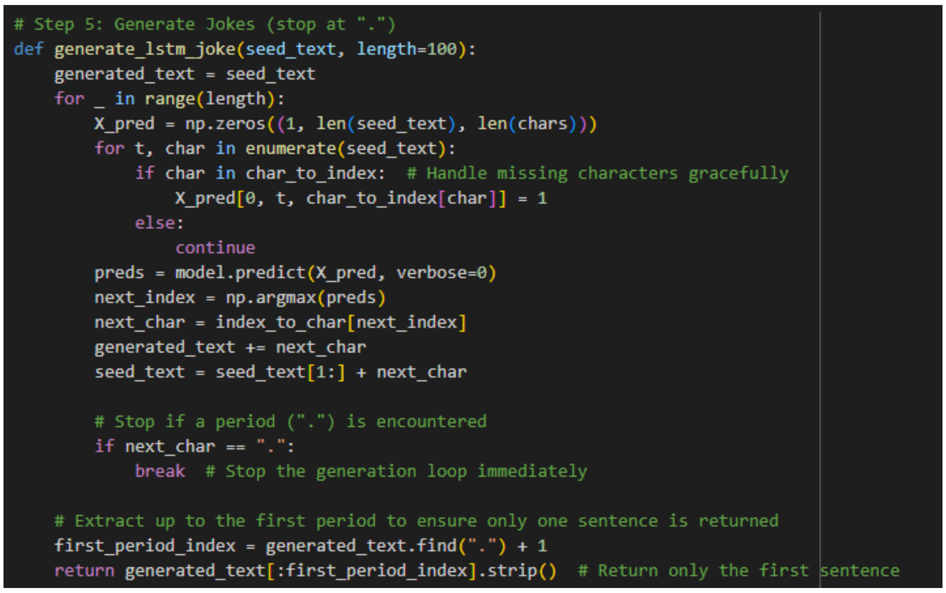
Next, for this part of code, it will load a previously trained joke generator model if it has the model, else it will build and train the joke generator model. If the model does not exist, the code will build a new model from scratch using the Keras Sequential class. This model is designed to process the joke data and predict the next character. There are multiple layers created to build the model. There is an LSTM layer with 128 units which will learn the pattern in sequence of the text. Then we have the Dropout layer with a 20% rate to reduce overfitting by randomly ignoring some neurons during training. Next, we have another LSTM layer and Dropout layer to further improve the generalization. Lastly, we have the Dense layer which uses a softmax activation function to predict the probability of each possible next character.

Then, the model is compiled with a categorical cross-entropy loss function which will measure how well the model predicts and the Adam optimizer with a learning rate of 0.001 which will adjust the model's weights to improve the accuracy.



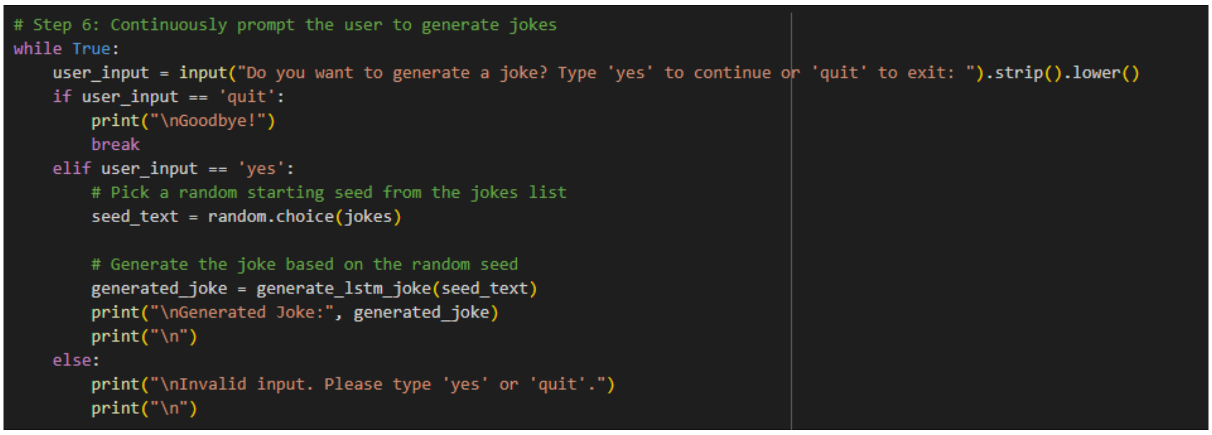
```
Epoch 1/20
52/52 ━━━━━━━━━━━━━━━━━━━━ 31s 536ms/step - loss: 3.2909
Epoch 2/20
52/52 ━━━━━━━━━━━━━━━━━━━━ 41s 535ms/step - loss: 2.9806
Epoch 3/20
52/52 ━━━━━━━━━━━━━━━━━━━━ 41s 535ms/step - loss: 2.9629
Epoch 4/20
52/52 ━━━━━━━━━━━━━━━━━━━━ 28s 533ms/step - loss: 2.8897
Epoch 5/20
52/52 ━━━━━━━━━━━━━━━━━━━━ 28s 536ms/step - loss: 2.7207
Epoch 6/20
52/52 ━━━━━━━━━━━━━━━━━━━━ 28s 537ms/step - loss: 2.4915
Epoch 7/20
52/52 ━━━━━━━━━━━━━━━━━━━━ 41s 541ms/step - loss: 2.3268
Epoch 8/20
52/52 ━━━━━━━━━━━━━━━━━━━━ 41s 538ms/step - loss: 2.1937
Epoch 9/20
52/52 ━━━━━━━━━━━━━━━━━━━━ 41s 540ms/step - loss: 2.0669
Epoch 10/20
52/52 ━━━━━━━━━━━━━━━━━━━━ 41s 539ms/step - loss: 1.9735
Epoch 11/20
52/52 ━━━━━━━━━━━━━━━━━━━━ 40s 534ms/step - loss: 1.9336
Epoch 12/20
52/52 ━━━━━━━━━━━━━━━━━━━━ 41s 532ms/step - loss: 1.8404
Epoch 13/20
52/52 ━━━━━━━━━━━━━━━━━━━━ 28s 535ms/step - loss: 1.7781
Epoch 14/20
52/52 ━━━━━━━━━━━━━━━━━━━━ 41s 531ms/step - loss: 1.7582
Epoch 15/20
52/52 ━━━━━━━━━━━━━━━━━━━━ 41s 533ms/step - loss: 1.6831
Epoch 16/20
52/52 ━━━━━━━━━━━━━━━━━━━━ 41s 532ms/step - loss: 1.6379
Epoch 17/20
52/52 ━━━━━━━━━━━━━━━━━━━━ 41s 536ms/step - loss: 1.6209
Epoch 18/20
52/52 ━━━━━━━━━━━━━━━━━━━━ 41s 548ms/step - loss: 1.5703
Epoch 19/20
52/52 ━━━━━━━━━━━━━━━━━━━━ 41s 542ms/step - loss: 1.5074
Epoch 20/20
52/52 ━━━━━━━━━━━━━━━━━━━━ 41s 542ms/step - loss: 1.4657
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file
Model trained and saved!
```

Once the model is created, it will be trained using the fit method, with the input data (x) and the target output (y). The training will be run in 20 epochs with a batch size of 128, which will process 128 sequences at a time. After done, it will save the model to the model_file so that it can be used again without the need to retrain.



```python
# Step 5: Generate Jokes (stop at ".")
def generate_lstm_joke(seed_text, length=100):
    generated_text = seed_text
    for _ in range(length):
        X_pred = np.zeros((1, len(seed_text), len(chars)))
        for t, char in enumerate(seed_text):
            if char in char_to_index:  # Handle missing characters gracefully
                X_pred[0, t, char_to_index[char]] = 1
            else:
                continue
        preds = model.predict(X_pred, verbose=0)
        next_index = np.argmax(preds)
        next_char = index_to_char[next_index]
        generated_text += next_char
        seed_text = seed_text[1:] + next_char

        # Stop if a period (".") is encountered
        if next_char == ".":
            break  # Stop the generation loop immediately

    # Extract up to the first period to ensure only one sentence is returned
    first_period_index = generated_text.find(".") + 1
    return generated_text[:first_period_index].strip()  # Return only the first sentence
```

This part of the code defines the generate_lstm_jokes function, which will create jokes using the trained LSTM model. First, it will provide the seed_text as input by converting each character into a one-hot encoded format. By using the model.predict function, the model generates the next character by analyzing the current sequence of text. The character with the highest probability is selected and added to the generated joke. The function then updates the seed_text by shifting it to include the new character, preparing it for the next prediction. It will continue to predict until it reaches "." is predicted. It will be the end of the jokes generated. To ensure that the output is a complete sentence, the function trims the generated text up to and including the first period and returns it as concise and coherent.

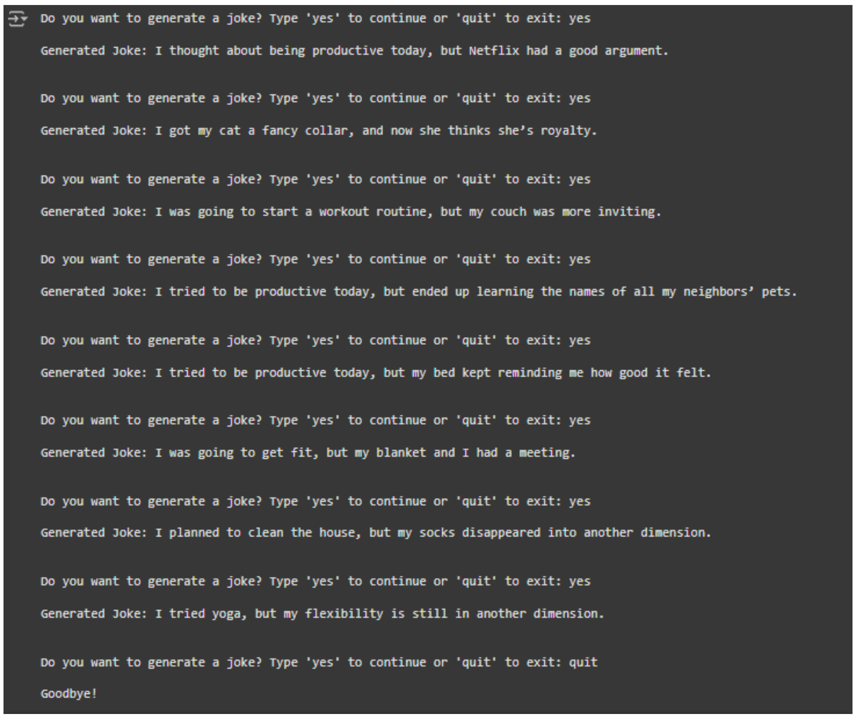


```python
# Step 6: Continuously prompt the user to generate jokes
while True:
    user_input = input("Do you want to generate a joke? Type 'yes' to continue or 'quit' to exit: ").strip().lower()
    if user_input == 'quit':
        print("\nGoodbye!")
        break
    elif user_input == 'yes':
        # Pick a random starting seed from the jokes list
        seed_text = random.choice(jokes)

        # Generate the joke based on the random seed
        generated_joke = generate_lstm_joke(seed_text)
        print("\nGenerated Joke:", generated_joke)
        print("\n")
    else:
        print("\nInvalid input. Please type 'yes' or 'quit'.")
        print("\n")
```

This is the last part of the code, it will allow the program to interact with the user and generate jokes continuously until the user decides to stop.

Results

<u>Generated Jokes Output:</u>



```
Do you want to generate a joke? Type 'yes' to continue or 'quit' to exit: yes

Generated Joke: I thought about being productive today, but Netflix had a good argument.


Do you want to generate a joke? Type 'yes' to continue or 'quit' to exit: yes

Generated Joke: I got my cat a fancy collar, and now she thinks she's royalty.


Do you want to generate a joke? Type 'yes' to continue or 'quit' to exit: yes

Generated Joke: I was going to start a workout routine, but my couch was more inviting.


Do you want to generate a joke? Type 'yes' to continue or 'quit' to exit: yes

Generated Joke: I tried to be productive today, but ended up learning the names of all my neighbors' pets.


Do you want to generate a joke? Type 'yes' to continue or 'quit' to exit: yes

Generated Joke: I tried to be productive today, but my bed kept reminding me how good it felt.


Do you want to generate a joke? Type 'yes' to continue or 'quit' to exit: yes

Generated Joke: I was going to get fit, but my blanket and I had a meeting.


Do you want to generate a joke? Type 'yes' to continue or 'quit' to exit: yes

Generated Joke: I planned to clean the house, but my socks disappeared into another dimension.


Do you want to generate a joke? Type 'yes' to continue or 'quit' to exit: yes

Generated Joke: I tried yoga, but my flexibility is still in another dimension.


Do you want to generate a joke? Type 'yes' to continue or 'quit' to exit: quit

Goodbye!
```

<u>Jokes Statistics</u>

	Jokes
count	99.000000
mean	68.808081
std	9.453082
min	48.000000
25%	60.000000
50%	64.000000
75%	72.000000
max	96.000000

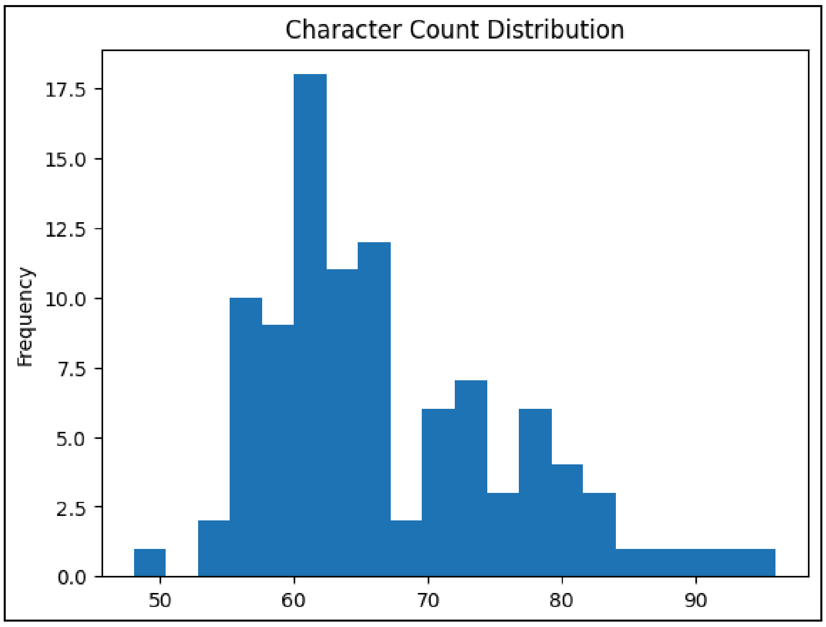
General descriptive statistics summarized the overview of the dataset which is comprehensive and easier to document the structure of the jokes generator. Adam optimizers modeling has trained the dataset to generate the jokes generator in text-based form. This table shows there are 99 jokes in the dataset. Besides that, metrics such as mean, median, and standard deviation of the jokes were calculated to know the pattern and the variation of the jokes. It also shows the minimum and maximum length of the jokes by calculating them by the characters or words.

<u>Jokes Details</u>

Index	**Word_Count**	**Char_Count**
0	13	66
1	16	83
2	13	58
3	16	86
4	13	62

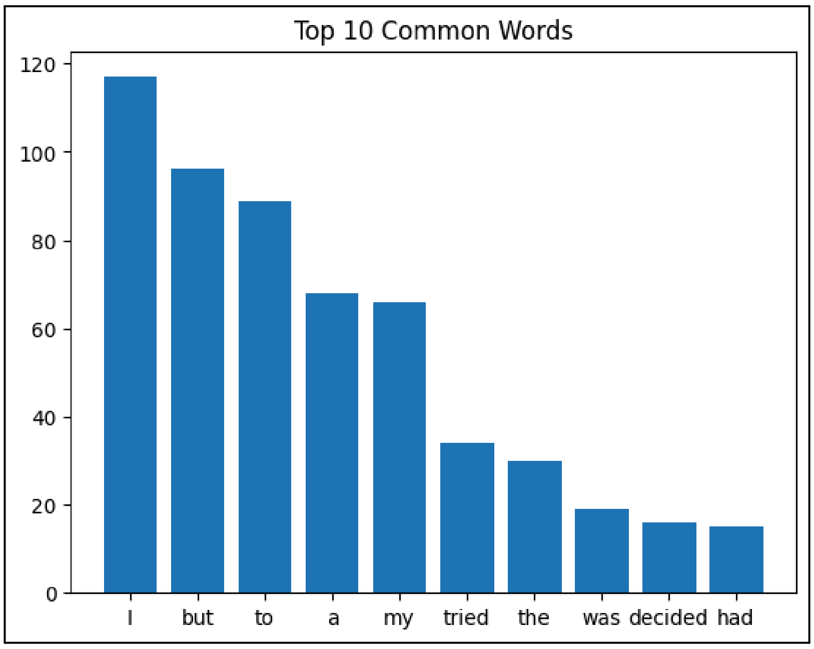
This table shows the details of the jokes to present the total words and characters used by every joke. The complexity of the jokes will be known from this table. From this table, more insight such as whether the structure or punchlines of the joke are short, long, or moderate.

Character Count Distribution



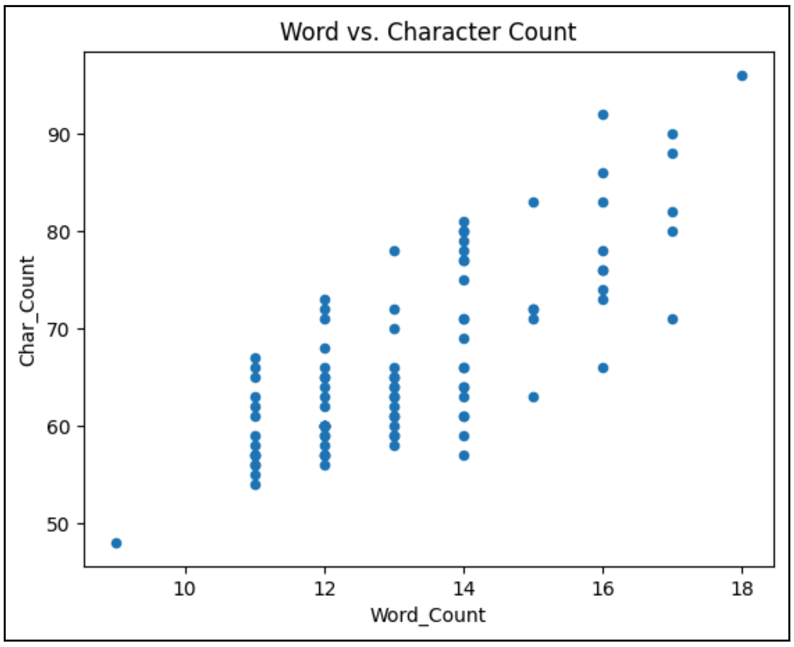
This histogram shows that most of the text entries, such as jokes are concise with the majority falling in the mid-range of character counts (60-70). Very short or very long entries are less common. This consistency might reflect an international effort to keep the text concise and engaging, which is typical for humor.

Top 10 Common Words



This histogram shows the top 10 common words that we have from the dataset that we use to generate the joke generator. The highest common word that we can see from the histogram is "I". Then, we have "but" as the second place and "to" as the third highest.

Word vs Character Count



Analysis of Result

a. General Description of Results

The generated jokes provide a consistent theme centered on humor derived from the common challenges of staying healthy, productive and motivated. Each joke shows a scenario where someone plans to complete a task but encounters an exaggerated obstacle that prevents them from doing it. This approach effectively captures relatable experiences such as laziness and distraction that make the jokes easy to understand and relatable. The humor comes from taking everyday distractions or challenges and turning them into funny stories that almost anyone can enjoy. For example, the joke might highlight how someone wants to do an exercise but there is something that is blocking that person from doing so. This kind of humor is something that always happens in our daily life that makes people see the absurdity in their own struggles.

Furthermore, the jokes manage to create a balance between relatability and creativity. This will keep the jokes engaging while maintaining a light-hearted one. This kind of joke is very important and great to have as it doesn’t focus on funny things only but also focuses on something that can make us feel emotional with the jokes.

b. Identifying Cases

There are some cases that we can get from the results:

Case	**Example**	**Reason**
Best Case (Most Creative)	"I planned to clean the house, but my socks disappeared into another dimension."	This joke is unique, blending an everyday task with a fantastical scenario.
Worst Case (Least Engaging)	"I tried to be productive today, but my bed kept reminding me how good it felt."	Although relatable, it lacks creativity and feels predictable.

Optimum Case (Balanced Creativity and Relatability)	"I was going to get fit, but my blanket and I had a meeting."	This joke is a balance between being relatable and creative.

There are three cases while generating this joke generator which is best case, worst case and optimum case. The first case is the best case which can predict the creative pattern of the jokes that have a sense of humor because they do not have a negative impact on anyone. It is effective because the jokes can make people laugh. At the same time, people can release their stress by having the best case of the jokes.

Next, the worst case is because of a lack of humor and overfitting to the sentence structures. The jokes feel like they are very predictable, lacking originality and failing to surprise the audience. These jokes are commonly known as cliche because of it always being repetitive in many ways. This highlights the importance of maintaining the balance between relatability and creativity in jokes generation.

The last one is the optimum case. This is the kind of joke that manages to balance between relatability and creativity that makes us feel very engaged with it. This balanced approach ensures that the jokes will be very effective in entertaining the audience and sustain their interest.

c. General Comparison of Scenarios

These are some general comparison from the results:

Scenario	**Creativity**	**Relatibility**	**Humor Impact**
Netflix vs productivity	Moderate	High	Moderate
Cat thinks she's royalty	High	Moderate	High
Couch vs workout routine	Moderate	High	Moderate
Learned neighbor's pets names	High	Moderate	High
Bed reminding how good it felt	Low	High	Low
Blanket and I had a meeting	High	High	High
Socks disappeared into dimension	High	Moderate	High
Flexibility in another dimension	Moderate	Moderate	Moderate

Conclusion

The objective of the project was to create a joke generator powered by AI that utilizes Long Short-Term Memory (LSTM) networks optimized through the Adam optimizer. The emphasis was on grasping humor trends and improving human-AI interaction by generating humor. A dataset of 99 self-created jokes was utilized, transformed into numerical format for training the LSTM model. The architecture of the model comprised LSTM layers, Dropout layers to avoid overfitting, and a Dense layer featuring a softmax activation function. The Adam optimizer was utilized to reduce the categorical cross-entropy loss, and the model underwent training for 20 epochs with a batch size of 128.

The findings indicated that the model effectively created coherent and funny jokes. The collection contained 99 jokes, with an average length of 68.8 characters. An examination of the jokes indicated trends in both word and character numbers, with the majority being brief and typically between 60 and 70 characters long. The jokes predominantly featured the words "I," "but," and "to." The produced jokes were typically relatable and imaginative, frequently emphasizing daily struggles in a humorous way. The analysis revealed three categories of cases: the top cases were imaginative and funny, the lowest cases were expected and less captivating, while the ideal cases struck a balance between creativity and relatability.

Suggestions for upcoming projects involve boosting the originality of the jokes, increasing the dataset to elevate model effectiveness, and creating an interactive platform where users can create and evaluate jokes, offering feedback for additional model enhancement. Future developments might investigate hybrid approaches that merge LSTM with different neural networks for enhanced effectiveness, include contextual data to create more pertinent and captivating jokes, and tackle possible biases in joke creation to guarantee inclusivity and suitability. This project aids in grasping humor creation in AI and provides practical uses in the entertainment sector, improving user interaction with AI via humor.

References

Chippada, B., & Saha, S. (2018). Knowledge Amalgam: Generating Jokes and Quotes Together. http://arxiv.org/abs/1806.04387

Faulkner, R., Rex, R., & Ryan, M. (n.d.). Knock Knock: Neural Joke Generation and Classification Final Report. https://github.com/Richard2926/NLP-Final

Gabbard, J., & Miller, D. (2018). 8.0851 Project: Machine Learning from Scratch: Stochastic Gradient Descent and Adam Optimizer.

IEEE Xplore Full-Text PDF_. (n.d.).

Kazi, T., Joshi, S., Kaitharath, S., & Mirza, I. A. (2021). Transformer based Neural Joke Generator. In International Journal of Computer Applications (Vol. 183, Issue 34).

Kingma, D. P., & Ba, J. (2014). Adam: A Method for Stochastic Optimization. http://arxiv.org/abs/1412.6980

quanhphan. (2024, April 30). Non-offensive Joke Generator. Kaggle.com; Kaggle. https://www.kaggle.com/code/quanhphan/non-offensive-joke-generator

Staudemeyer, R. C., & Morris, E. R. (2019). Understanding LSTM -- a tutorial into Long Short-Term Memory Recurrent Neural Networks. http://arxiv.org/abs/1909.09586

Suljic, V., & Pervan, A. (2024). Creative writing in the hands of artificial intelligence: the analysis of humour in Bard-generated texts. European Journal of Humour Research, 12(4), 251–268. https://doi.org/10.7592/EJHR2024.12.4.928

Ting-yu YEH Nicholas FONG Nathan KERR Brian COX Supervisor Ming-Hwa WANG, A. (2018). Generating Original Jokes.

Tiwari, D., Toady, T., Monperrus, M., & Baudry, B. (2024). With Great Humor Comes Great Developer Engagement. Proceedings - International Conference on Software Engineering, 1–11. https://doi.org/10.1145/3639475.3640099

Toplyn, J. (n.d.). Witscript 2: A System for Generating Improvised Jokes Without Wordplay. https://openai.com/api/

van Houdt, G., Mosquera, C., & Nápoles, G. (2020). A review on the long short-term memory model. Artificial Intelligence Review, 53(8), 5929–5955. https://doi.org/10.1007/s10462-020-09838-1

quanhphan. (2024, April 30). Non-offensive Joke Generator. Kaggle.com; Kaggle.

Winters, T., Nys, V., & Schreye, D. D. (2018). Automatic Joke Generation: Learning Humor from Examples. Lecture Notes in Computer Science, 360–377. https://doi.org/10.1007/978-3-319-91131-1_28

Chaudhary, T., Goel, M., & Mamidi, R. (2021). Towards Conversational Humor Analysis and Design. ArXiv.org. https://arxiv.org/abs/2103.00536

Appendix

We also put all these files in an appendix folder to make it is easy to open

Data File (Book1.xlsx)

Jokes
I spilled coffee on my laptop, and now it's trying to brew itself.
I tried to make a salad, but ended up eating the dressing straight from the bottle.
My dog learned to fetch, but only when I don't ask him to.
I bought a plant for the office, but it's the one thing that's actually thriving here.
I got my cat a fancy collar, and now she thinks she's royalty.
I once tried to be a morning person, but my bed had other plans.
I joined a gym, but the only weight I lost was my will to go back.
I tried to take a nap, but my thoughts decided to throw a party instead.
I walked into a room and forgot why I was there, which probably means I'm doing something right.
I asked for a sign, but the universe sent me a spam email instead.
I started a diet, but my fridge is a really bad influence.
I tried to write a novel, but all I wrote was a grocery list.
I went to the beach, but my sunscreen had other plans.
I bought a book on procrastination, but I'm too busy to read it.
I tried to stay awake, but my pillow was way too persuasive.
I decided to cut down on sugar, but cookies had other ideas.
I started organizing my closet, but the chaos outside the closet started to make more sense.
I tried to be productive today, but ended up learning the names of all my neighbors' pets.
I was going to clean my house, but then I realized it's more fun to watch Netflix.
I tried to be a morning person, but the morning wasn't having it.
I once tried to follow a recipe, but ended up inventing a new form of chaos.
I tried to meditate, but my brain had a meeting instead.
I made a to-do list, but somehow ended up making another list instead.
I went to the store to buy eggs, but left with a plant and a new hobby.
I went for a jog today, but my legs quickly voted against it.
I tried a new hobby, but it turns out I'm a professional procrastinator.
I thought about eating healthy, but then pizza called my name.
I got a new alarm clock, but now it just competes with my snooze button.
I tried to eat a salad, but the dressing turned it into a soup.
I tried yoga, but my flexibility is still in another dimension.
I tried to be productive today, but my bed kept reminding me how good it felt.
I decided to declutter my life, but my clutter had other ideas.

I planned to write an email, but I ended up watching cat videos for an hour.
I tried to cook dinner, but my kitchen staged a mutiny.
I decided to take a walk, but my feet insisted on staying at home.
I tried to organize my thoughts, but they scattered like confetti.
I went for a run, but it turned into a slow-paced stroll.
I planned to go for a walk, but then I remembered how cozy my blanket is.
I tried to make a smoothie, but it ended up more like a soup.
I went for a jog, but my feet quickly filed a complaint.
I tried to be productive, but my couch was more convincing.
I tried to be healthy today, but the donuts were calling.
I went to the gym, but they were out of motivational quotes.
I decided to exercise, but my motivation was still in bed.
I tried to eat healthy, but pizza had a strong argument.
I tried to take a nap, but my mind wanted to discuss life goals.
I started doing yoga, but my flexibility is still stuck in 2019.
I decided to eat clean, but then fries happened.
I planned to be productive, but my couch had other ideas.
I went to the store for apples, but ended up buying cookies.
I was going to start a workout routine, but my couch was more inviting.
I thought about organizing my closet, but I got distracted by my sock collection.
I tried to declutter, but my clutter was emotionally attached to me.
I tried to start a hobby, but procrastination beat me to it.
I thought about being productive today, but Netflix had a good argument.
I tried to make breakfast, but I accidentally made a mess instead.
I planned to write a report, but my dog needed emotional support.
I tried cooking dinner, but the recipe turned into a crime scene.
I decided to take a nap, but my dreams turned into meetings.
I thought about working out, but the couch and I reached a mutual understanding.
I decided to reorganize my life, but my chaos had other plans.
I planned to take a walk, but then I remembered how cozy my blanket is.
I decided to eat healthy, but the ice cream truck showed up.
I planned to go for a jog, but my energy filed for unemployment.
I went to buy groceries, but ended up buying snacks for the entire neighborhood.
I tried to learn a new skill, but my phone kept distracting me.
I wanted to go to the gym, but I couldn't find my motivation.
I planned to take a nap, but my thoughts had other plans.
I went to make a salad, but ended up ordering pizza instead.

I was going to get fit, but my blanket and I had a meeting.
I decided to get serious about exercising, but my bed was just too persuasive.
I tried to be healthy, but my cravings were in charge today.
I tried to write a book, but ended up writing a to-do list.
I started organizing my desk, but then I organized my thoughts instead.
I tried to eat vegetables, but they were overruled by cheese.
I planned to clean the house, but my socks disappeared into another dimension.
I tried to learn how to cook, but my microwave had other ideas.
I decided to start a new hobby, but my couch was just so comfortable.
I thought about going for a walk, but then my pillow won.
I went to clean the kitchen, but my fridge wanted to keep things interesting.
I tried to be productive, but my phone turned into a rabbit hole.
I was going to start a workout routine, but my blanket wouldn't let me.
I thought about going to the gym, but then the gym started feeling like a distant dream.
I was going to cook dinner, but then I remembered how easy it is to order pizza.
I tried to write a poem, but it turned into a grocery list.
I planned to organize my life, but I ended up organizing my snacks instead.
I was going to take a walk, but my thoughts decided to take a nap instead.
I decided to eat healthier, but chocolate insisted on being part of the plan.
I was going to start my day, but my bed asked me to stay.
I tried to be productive, but my sofa kept calling me back.
I decided to exercise, but the couch had me in a chokehold.
I thought about cleaning the house, but then my thoughts scattered.
I decided to stop procrastinating, but then I remembered I had a nap scheduled.
I thought about cooking, but my delivery app was more persuasive.
I was going to go for a run, but my legs are still on vacation.
I planned to organize my life, but my bed had a better idea.
I tried to be productive, but the internet kept offering me distractions.
I tried to eat healthy, but my fridge kept offering cake.
I decided to be productive, but I spent an hour finding the perfect playlist first.

Code File and Computational & Analysis File (Joke_Generator_with_Model.ipynb) - using google collab

To run this code, import Book1.xlsx and joke.generation.model.h5(in the appendix folder) files into Google Collab and run them one by one to ensure the generated joke is functional.

CODE

```
import pandas as pd

import numpy as np

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import LSTM, Dense, Dropout, Embedding

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.utils import get_file

import random


# Step 1: Load Jokes from Excel File

file_path = '/content/Book1.xlsx'

df = pd.read_excel(file_path)

# Assuming your Excel sheet has a column 'joke'

jokes = df['Jokes'].dropna().tolist()


# Step 2: Preprocess Data

# Join all jokes together to create a single string of jokes

text = " ".join(jokes)

# Create a set of all unique characters

chars = sorted(list(set(text)))

char_to_index = {char: index for index, char in enumerate(chars)}

index_to_char = {index: char for index, char in enumerate(chars)}

# Convert text to numerical data (index of each character)

sequence_length = 100

sequences = []

next_chars = []

for i in range(0, len(text) - sequence_length, 1):

    sequences.append(text[i:i + sequence_length])
    next_chars.append(text[i + sequence_length])

# Convert sequences to numerical data
X = np.zeros((len(sequences), sequence_length, len(chars)), dtype=bool)
y = np.zeros((len(sequences), len(chars)), dtype=bool)
for i, sequence in enumerate(sequences):
    for t, char in enumerate(sequence):
        X[i, t, char_to_index[char]] = 1
    y[i, char_to_index[next_chars[i]]] = 1

# Step 3: Build the LSTM Model
model = Sequential()
model.add(LSTM(128, input_shape=(X.shape[1], X.shape[2]), return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(128))
model.add(Dropout(0.2))
model.add(Dense(len(chars), activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer=Adam(learning_rate=0.001))

# Step 4: Train the Model
model.fit(X, y, batch_size=128, epochs=20)

# Step 5: Generate Jokes (stop at ".")
def generate_lstm_joke(seed_text, length=100):
    generated_text = seed_text
    for _ in range(length):
        X_pred = np.zeros((1, len(seed_text), len(chars)))
        for t, char in enumerate(seed_text):
            if char in char_to_index:  # Handle missing characters gracefully
                X_pred[0, t, char_to_index[char]] = 1
            else:
                continue
        preds = model.predict(X_pred, verbose=0)
        next_index = np.argmax(preds)
        next_char = index_to_char[next_index]
        generated_text += next_char
        seed_text = seed_text[1:] + next_char
        # Stop if a period (".") is encountered
        if next_char == ".":
            break  # Stop the generation loop immediately
    # Extract up to the first period to ensure only one sentence is returned
    first_period_index = generated_text.find(".") + 1
    return generated_text[:first_period_index].strip()  # Return only the first sentence


# Step 6: Continuously prompt the user to generate jokes
while True:
    user_input = input("Do you want to generate a joke? Type 'yes' to continue or 'quit' to exit: 
").strip().lower()
    if user_input == 'quit':
        print("\nGoodbye!")
        break
    elif user_input == 'yes':
        # Pick a random starting seed from the jokes list
        seed_text = random.choice(jokes)
        # Generate the joke based on the random seed
        generated_joke = generate_lstm_joke(seed_text)
        print("\nGenerated Joke:", generated_joke)
        print("\n")
    else:
        print("\nInvalid input. Please type 'yes' or 'quit'.")
        print("\n")
```

<u>OUTPUT</u>

```
Do you want to generate a joke? Type 'yes' to continue or 'quit' to exit: yes
Generated Joke: I thought about being productive today, but Netflix had a good argument.
Do you want to generate a joke? Type 'yes' to continue or 'quit' to exit: yes
Generated Joke: I got my cat a fancy collar, and now she thinks she's royalty.
Do you want to generate a joke? Type 'yes' to continue or 'quit' to exit: yes
Generated Joke: I was going to start a workout routine, but my couch was more inviting.
Do you want to generate a joke? Type 'yes' to continue or 'quit to exit: yes
Generated Joke: I tried to be productive today, but ended up learning the names of all my neighbors' pets.
Do you want to generate a joke? Type 'yes' to continue or 'quit' to exit: yes
Generated Joke: I tried to be productive today, but my bed kept reminding me how good it felt.
Do you want to generate a joke? Type 'yes' to continue or 'quit' to exit: yes
Generated Joke: I was going to get fit, but my blanket and I had a meeting.
Do you want to generate a joke? Type 'yes' to continue or 'quit* to exit: yes
Generated Joke: I planned to clean the house, but my socks disappeared into another dimension.
Do you want to generate a joke? Type 'yes' to continue or 'quit' to exit: yes
Generated Joke: I tried yoga, but my flexibility is still in another dimension.
Do you want to generate a joke? Type 'yes' to continue or 'quit' to exit: quit
Goodbye!
```

Computation & Analysis File (Joke_Generator_with_Model.ipynb) - using google collab

<u>CODE</u>

```
#step 7
df['Jokes'].describe()
df['Jokes'].str.len().describe()  # Length of each joke
```

```
#step 8

df['Word_Count'] = df['Jokes'].str.split().str.len()

df['Char_Count'] = df['Jokes'].str.len()

import matplotlib.pyplot as plt

df['Char_Count'].plot(kind='hist', bins=20, title='Character Count Distribution')

plt.show()


#step 9

from collections import Counter

words = ' '.join(df['Jokes']).split()

common_words = Counter(words).most_common(10)

words, counts = zip(*common_words)

plt.bar(words, counts)

plt.title('Top 10 Common Words')

plt.show()


#step 10

from wordcloud import WordCloud

text = ' '.join(df['Jokes'])

wordcloud = WordCloud().generate(text)

plt.imshow(wordcloud, interpolation='bilinear')

plt.axis('off')

plt.show()


#step 11

df.plot(x='Word_Count', y='Char_Count', kind='scatter', title='Word vs. Character Count')

plt.show()
```