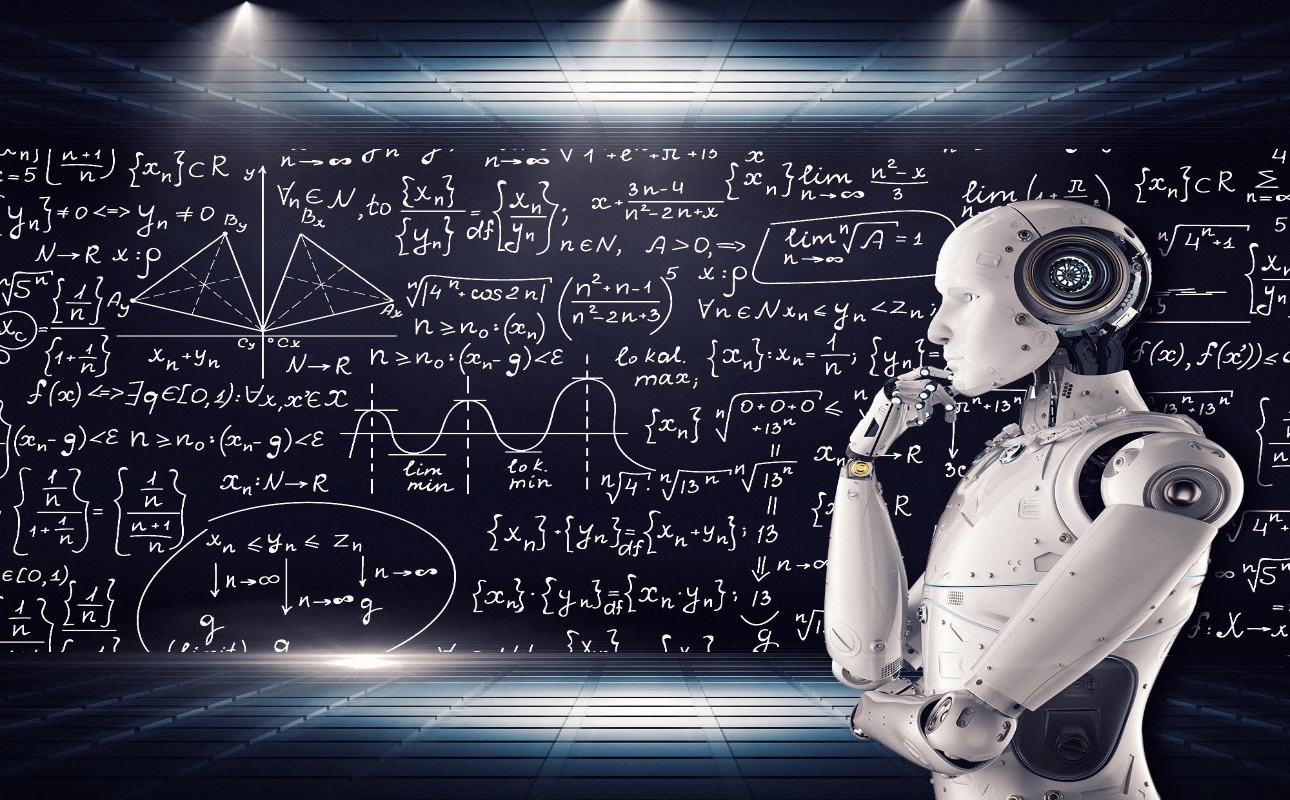
## P1#yIS1



# 

# AAI-520: **Advanced Generative Chatbot Design**

(Using cornell movie dialogs corpus)

October 2023

Group 6: Paul Parks, Alden Caterio, Adam Graves

Github Link to Model: https://github.com/agraves13/AAI520

Table of Contents

[P1#yIS1 0](#_Toc146110627)

[AAI-520: Advanced Generative Chatbot Design 0](#_Toc146110628)

[Abstract 2](#_Toc146110629)

[Goals/Strategies 2](#_Toc146110630)

[Processing Composer Data into a DataFrame 2](#_Toc146110631)

[Data Fields of the generated dataset 2](#_Toc146110632)

[Model Architecture 2](#_Toc146110633)

[LSTM Models 2](#_Toc146110634)

[Model Architecture 2](#_Toc146110635)

[Results Analysis 2](#_Toc146110636)

[Appendix A: 3](#_Toc146110637)

[Code Implementation Details 3](#_Toc146110638)

[References 8](#_Toc146110639)

# Abstract

This system is a Chatbot developed utilizing Natural language processing (NLP) services. NLP is a field of computer science that deals with the interaction between computers and human (natural) languages. It's used in a wide range of applications, including machine translation, speech recognition, and text analysis.

Chatbots are computer programs that can simulate conversation with humans. They can be used in a variety of settings, including customer service, education, and entertainment.

This project aims to develop an NLP-based chatbot that can provide information about movies from the Cornell Movie-Dialog Corpus (See: https://www.kaggle.com/datasets/rajathmc/cornell-moviedialog-corpus). The chatbot will be trained on a dataset of movie reviews and ratings from IMDB. This will allow the chatbot to understand the nuances of human language and to provide accurate and informative responses to user queries.

The chatbot will be able to answer a variety of questions about movies, such as:

* What is the title of a movie?
* Who are the actors and actresses in the movie?
* What is the rating of the movie?
* What year did the movie come out?
* What is the genre of the movies?

The chatbot will also be able to generate recommendations for movies based on the user's preferences. For example, if the user says that they like science fiction movies, the chatbot can recommend a list of science fiction movies that are highly rated and have positive reviews.

The chatbot will be developed using the following NLP technologies:

* Natural language understanding (NLU): NLU is a field of NLP that deals with the ability of computers to understand the meaning of human language. The chatbot will use NLU to understand the meaning of the user's queries and to generate informative responses.
* Natural language generation (NLG): NLG is a field of NLP that deals with the ability of computers to generate human-like text. The chatbot will use NLG to generate responses to the user's queries in a clear and concise manner.

The chatbot will be evaluated on its ability to provide accurate and informative responses to user queries, and on its ability to generate recommendations that are relevant to the user's preferences.

# Goals/Strategies

**Goal**: Build a user friendly chatbot that can carry out multi-turn conversations, adapt to context, and handle a variety of questions about movies.

**Strategy Phase**

**Step 1**

Define the Objectives and Use Cases:

This chatbot will communicate with any user that wishes to inquire into information pertaining to a movie. This will include any information that is available on the loaded database. Any information not available will answer with an apology error.

**Step 2**

Platform and Technology:

The chatbot will be a front-end GUI allowing the user to enter a question and receive a response. Activated by Enter button or a Send button.

**Step 3**

Perform the Extract, Transform, & Load (ETL) process in which we will have a full dataset. In this dataset we will have:

A corpus of five files that contains a metadata-rich collection of fictional conversations extracted from raw movie scripts:

- 220,579 conversational exchanges between 10,292 pairs of movie characters

- involves 9,035 characters from 617 movies

- in total 304,713 utterances

- movie metadata included:

- genres

- release year

- IMDB rating

- number of IMDB votes

- IMDB rating

- character metadata included:

- gender (for 3,774 characters)

- position on movie credits (3,321 characters)

- movie\_titles\_metadata.txt

- contains information about each movie title

- fields:

- movieID,

- movie title,

- movie year,

- IMDB rating,

- no. IMDB votes,

- genres in the format ['genre1','genre2', ,'genreN']

- movie\_characters\_metadata.txt

- contains information about each movie character

- fields:

- characterID

- character name

- movieID

- movie title

- gender ("?" for unlabeled cases)

- position in credits ("?" for unlabeled cases)

- movie\_lines.txt

- contains the actual text of each utterance

- fields:

- lineID

- characterID (who uttered this phrase)

- movieID

- character name

- text of the utterance

- movie\_conversations.txt

- the structure of the conversations

- fields

- characterID of the first character involved in the conversation

- characterID of the second character involved in the conversation

- movieID of the movie in which the conversation occurred

- list of the utterances that make the conversation, in chronological

order: ['lineID1','lineID2', ,'lineIDN']

has to be matched with movie\_lines.txt to reconstruct the actual content

- raw\_script\_urls.txt

- the urls from which the raw sources were retrieved

**Note: The five files are attached to rows by the Movie\_ID field.**

**Step 4**

Clean up the data, verify NaN, and any bad data. Verify field types

**Step 5**

Natural Language Processing (NLP) and Understanding:

* Process Tokenizing to token the input words
* Use Word2Vec to perform the semantic analysis which will assist in identifying the intention of the question and help correlate movie information to multiple movies that can be part of the answer/reply.
* Dialogue Management Systems: Possible to look at library Rasa which is designed specifically for building conversational AI and managing dialogue context effectively.
* Build and Train the Chatbot: Train the chatbot using datasets and fine-tuning methods to improve its conversational abilities.
* Test and revisions: Test the system and modify as needed.

**Step 6**

Complete documentation and knowledge base

**Step 7**

Execute and release the chatbot to users

**Step 8**

Create presentation as noted

# Design

# Create a Colab Jupyter Notebook and share to the team

# Load the Cornell Movie Database via:

|  |  |  |
| --- | --- | --- |
| 0BIMPORTING LIBRARIES | 1BImporting the necessary libraries to read the dataset: Most are standard to the work done in this class regarding deep machine learning. The new ones that are loading deal with the MIDI file formatting: they are: | **import** os  **import** glob  **import** pretty\_midi  **import** numpy **as** np  **import** matplotlib.pyplot **as** plt  **import** pandas **as** pd  *# Ignore warnings*  **import** warnings  warnings**.**filterwarnings('ignore')  **from** sklearn.metrics **import** accuracy\_score, precision\_score, recall\_score, f1\_score  **import** tensorflow **as** tf  **from** tensorflow.keras **import** layers, models, optimizers  **from** tensorflow.keras.preprocessing.image **import** ImageDataGenerator  **from** sklearn.metrics **import** classification\_report  **from** keras **import** models, layers  *# Check if pickle file exists and use the file for dataset*  **import** pickle |
| 2BData Processing Functions | 3BFunctions created to iterate through composer midi files and create dataset. | #@title 2.1: Extract features using librosa for further feature extraction  def calculate\_features(midi\_file):  # Load MIDI file  midi\_data = pretty\_midi.PrettyMIDI(midi\_file)  # Time interval for calculating features  interval = 1.0 # 1 second  times = np.arange(0, midi\_data.get\_end\_time(), interval)  # Create arrays for storing time series data  pitch = np.zeros(len(times))  volume = np.zeros(len(times))  note\_density = np.zeros(len(times))  tempo = np.zeros(len(times))  # Calculate time series data for each feature  for i, t in enumerate(times):  # Get notes that are playing at this time  notes = [note for note in midi\_data.instruments[0].notes if note.start <= t < note.end]  # Calculate average pitch  if notes:  pitch[i] = np.mean([note.pitch for note in notes])  # Calculate note density (notes per second)  note\_density[i] = len(notes) / interval  # Calculate average volume  if notes:  volume[i] = np.mean([note.velocity for note in notes])  # Calculate rhythmic complexity (variance in inter-onset intervals)  inter\_onset\_intervals = np.diff([note.start for note in midi\_data.instruments[0].notes])  rhythmic\_complexity = np.var(inter\_onset\_intervals)  # Calculate tempo for each moment in time  tempo\_changes = midi\_data.get\_tempo\_changes()  tempo = np.interp(times, tempo\_changes[0], tempo\_changes[1])  return times, pitch, note\_density, volume, rhythmic\_complexity, tempo  #@title 2.2: Process composer data to df  def process\_composer\_data():  # Initialize DataFrame  df = pd.DataFrame(columns=["Composer","Times", "Pitch", "Note\_Density", "Volume",  "Rhythmic\_Complexity", "Tempo"])  # Iterate over all composer directories  for composer\_dir in glob.glob(os.path.join(base\_dir, '\*')):  # Get the composer's name  composer\_name = os.path.basename(composer\_dir)  print(f"Processing {composer\_name} MIDI files...")  # Iterate over all MIDI files in composer's directory  for midi\_file in glob.glob(os.path.join(composer\_dir, '\*.mid')):  print(f"Processing {midi\_file}...")  try:  times, pitch, note\_density, volume, rhythmic\_complexity, tempo = calculate\_features(midi\_file)  # Append to DataFrame  df = df.append({"Composer": composer\_name, "Times": times, "Pitch": pitch,  "Note\_Density": note\_density, "Volume": volume,  "Rhythmic\_Complexity": rhythmic\_complexity,  "Tempo": tempo},  ignore\_index=True)  except Exception as e:  print(f"Error processing {midi\_file}: {str(e)}")  # Write the DataFrame to a pickle file  df.to\_pickle(base\_dir + "/" + pickle\_file\_name)  return df |
| 4BPickle File Use | Using pickle files to allow one teammate to generate a pickle file and allow of the team not to have repeat work | #@title 2.3: Data Processing - Feature extraction  pickle\_file = base\_dir + "/" + pickle\_file\_name  # Check if the pickle file exists  if not os.path.exists(pickle\_file):  print("Music Data not Pickled, creating dataset using feature extract.")  df = process\_composer\_data()  else:  # Open the pickle file in binary mode and load the data  with open(pickle\_file, 'rb') as file:  data = pickle.load(file)  # Create a DataFrame from the loaded data  df = pd.DataFrame(data)  # Now you have your DataFrame ready for use  print(df.head()) |
| 5BData Preparation | Preparing the data for LSTM  Split train and test data sets (80-20)  Transform the data for the LSTM | # Convert all other features to have an extra dimension for LSTM  def transform\_series(series, num\_steps):  # Reshape series to (samples, time\_steps, features)  X = np.zeros((len(series), num\_steps, 1))  for i in range(len(series)):  X[i,:,0] = series.iloc[i][:num\_steps]  return X  --------------------------------------------------------------------------------------------------  #@title 3.1: Split train and test data sets (80-20)  # Using stratify to ensure the datasets have same prorportions of each composer as original dataset  df\_train\_val, df\_test = train\_test\_split(df, test\_size=0.2, random\_state=42, stratify=df['Composer'])  # Second, we separate the remaining data into the train and validation sets (75-25)  df\_train, df\_val = train\_test\_split(df\_train\_val, test\_size=0.25, random\_state=42, stratify=df\_train\_val['Composer'])  # The train/val/test split is now 60%/20%/20%  # Encode the labels  encoder = LabelEncoder()  encoder.fit(df['Composer']) # Fit on the whole dataset  # Transform the labels to one-hot encoded form for each subset  y\_train = np\_utils.to\_categorical(encoder.transform(df\_train['Composer']))  y\_val = np\_utils.to\_categorical(encoder.transform(df\_val['Composer']))  y\_test = np\_utils.to\_categorical(encoder.transform(df\_test['Composer']))  --------------------------------------------------------------------------------------------------  #@title 3.2: Apply transform\_series on each feature for each subset  def prepare\_data(df, num\_steps):  pitch = transform\_series(df['Pitch'], num\_steps)  note\_density = transform\_series(df['Note\_Density'], num\_steps)  volume = transform\_series(df['Volume'], num\_steps)  rhythmic\_complexity = np.array([df['Rhythmic\_Complexity'].values]\*num\_steps).T[:,:,np.newaxis]  tempo = transform\_series(df['Tempo'], num\_steps)  X = np.concatenate([pitch, note\_density, volume, rhythmic\_complexity, tempo], axis=-1)  return X  num\_steps = 27  X\_train = prepare\_data(df\_train, num\_steps)  X\_val = prepare\_data(df\_val, num\_steps)  X\_test = prepare\_data(df\_test, num\_steps) |
| 6BResults Visualization | Plotting function to allow visualization of data.  Print the true class labels and predicted class labels  Plot  confusion Matrix  Print Classification Report | #@title 5: Plot the training and validation loss V1  def plot\_learning\_curves(history):  plt.plot(history1.history['loss'], label='Train Loss')  plt.plot(history1.history['val\_loss'], label='Validation Loss')  plt.legend()  plt.show()  # Plot the training and validation accuracy  plt.plot(history1.history['accuracy'], label='Train Accuracy')  plt.plot(history1.history['val\_accuracy'], label='Validation Accuracy')  plt.legend()  plt.show()  return plot\_learning\_curves  --------------------------------------------------------------------------------------------------  #@title 6.1v: Print the true class labels and predicted class labels  for true\_label, pred\_label in zip(y\_test, y\_test\_pred\_class):  true\_class = encoder.inverse\_transform([np.argmax(true\_label)])[0]  pred\_class = encoder.inverse\_transform([pred\_label])[0]  print(f"True Class: {true\_class}, Predicted Class: {pred\_class}")  --------------------------------------------------------------------------------------------------  #@title 6.2: Print a confusion matrix V1  from sklearn.metrics import confusion\_matrix  import seaborn as sns  #mport matplotlib.pyplot as plt  # Generate the confusion matrix  cm = confusion\_matrix(np.argmax(y\_test, axis=1), y\_test\_pred\_class)  # Create a heatmap of the confusion matrix  plt.figure(figsize=(8, 6))  sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=encoder.classes\_, yticklabels=encoder.classes\_)  plt.xlabel('Predicted Labels')  plt.ylabel('True Labels')  plt.title('Confusion Matrix')  plt.show()  --------------------------------------------------------------------------------------------------  #@title 6.3: Report the confusion matrix V1  # Reshape the input data for prediction  X\_test\_scaled\_reshaped = X\_test\_scaled.reshape(X\_test\_scaled.shape[0], num\_steps, num\_features)  # Make predictions on the reshaped test set  y\_test\_pred = model.predict(X\_test\_scaled\_reshaped)  y\_test\_pred\_class = np.argmax(y\_test\_pred, axis=1)  # Convert predicted class labels back to original composer labels using the encoder  lstm\_pred\_labels = encoder.inverse\_transform(y\_test\_pred\_class)  lstm\_true\_labels = encoder.inverse\_transform(np.argmax(y\_test, axis=1)) # Convert true class labels back  # Print the classification report  lstm\_classification\_report\_v1 = classification\_report(lstm\_true\_labels, lstm\_pred\_labels, target\_names=encoder.classes\_)  print("Classification Report - LSTM Model ver 1:\n", lstm\_classification\_report\_v1) |
| 7BSpectrogram generation Code | Convert midi files to wav files using fluidsynth  Generate png files from wav files using librosa | #!/bin/bash  # Set the location of your soundfont file  sound\_font="../TimGM6mb.sf2"  # Iterate over all directories in the current directory  for dir in \*/  do  # Go inside each directory  cd "$dir"  # Iterate over all .mid files in the current directory  for midi\_file in \*.mid  do  # Replace the file extension from .mid to .wav  wav\_file="${midi\_file%.mid}.wav"  # Use fluidsynth to convert the midi file to a wav file  /mnt/host/c/tools/fluidsynth-2.3.2-win10-x64/bin/fluidsynth.exe -ni "$sound\_font" "$midi\_file" -F "$wav\_file" -r 44100  done  # Go back to the parent directory  cd ..  done  -----------------------------------------------------------------------  def generate\_images(dataset\_path):  X = []  y = []  composers = os.listdir(dataset\_path)  for i, composer in enumerate(composers):  composer\_path = os.path.join(dataset\_path, composer)  # Check if it is a directory  if os.path.isdir(composer\_path):  for filename in os.listdir(composer\_path):  if filename.endswith('.wav'):  print(dataset\_path + "/" + composer + "/" + filename)  %matplotlib inline  x , sr = librosa.load(dataset\_path + "/" + composer + "/" + filename)  X = librosa.stft(x)  img\_filename = change\_extension(filename, ".png")  # Generate spectrogram  D = np.abs(X)  # Resize to 224x224  D\_resized = ndimage.zoom(D, (224.0/D.shape[0], 224.0/D.shape[1]))  # Generate the image  plt.figure(figsize=(5, 5))  librosa.display.specshow(librosa.amplitude\_to\_db(D\_resized, ref=np.max), sr=sr, x\_axis='time', y\_axis='log')  plt.tight\_layout()  plt.savefig(dataset\_path + "/" + composer + "/" + img\_filename)  print(img\_filename)  plt.close() |

# References

AWS Machine Learning Blog, taken from: <https://aws.amazon.com/blogs/machine-learning/reduce-deep-learning-training-time-and-cost-with-mosaicml-composer-on-aws/>)