Introduction:

The sorting of individuals into categories based on their personalities and traits has many applications, from psychology to literature, and serves as a tool for understanding human behavior and character dynamics. One such iconic sorting system is the Hogwarts Houses from the Harry Potter series, where people are sorted into Gryffindor, Hufflepuff, Ravenclaw, or Slytherin based on their unique traits and values.

While Harry Potter sorting quizzes are popular among fans, their conventional design and predictability often lead to biased outcomes, since individuals can easily select answers that align with their desired outcome rather than those that reflect their true personality. This raises concerns about the validity and accuracy of the results, leading to a need for more objective methods for sorting people into Hogwarts Houses, ensuring accuracy in the classification process.

**In response to this challenge,** this paper presents a unique approach for sorting Harry Potter characters based on their dialogue from the movie series using a stacked LSTM network. Our methodology aims to overcome the limitations of traditional sorting quizzes, providing a more objective and reliable means of categorizing characters into their respective Houses using the Harry Potter Movies dataset on Kaggle \cite{dataset}. We noticed that the original dataset is unbalanced, as Gryffindor characters have many more lines than characters from other houses. To combat this, we will run our model on 3 datasets after pre-processing the data:

* Regular: the original dataset
* Shortened: a dataset where we reduce the number of lines for houses with a lot of lines to match the number of lines of the house with the least number of lines
* Duplicate: a dataset where we increase the number of lines (by duplicating them) for houses with less lines to match the number of lines of the house with the most number of lines

Literature Review:

Many people have conducted research related to sentiment analysis and classification using various methods. The work of some such papers is summarized below.

**Study of Dependency on Number of LSTM Units for Character-based Text Generation Models**

S. Chakraborty et. al. \cite{carl\_1} studied the effect of the number of LSTM cells on the performance of a generative model using the C programming language. They used two models that are fed the same input: one with 2 LSTM layers followed by a dense layer, and another model with an added Convolutional layer before the LSTM layers to do feature extraction. They found that accuracy increased and loss decreased as the number of LSTM cells increased, but started overfitting at 192 LSTM cells, as well as that the loss and accuracy with a large number of LSTM cells were better with the added Convolutional layer.

**Efficient Method for Personality Prediction using Hybrid Method of Convolutional Neural Network and LSTM**

N. Sujatha et. al. \cite{carl\_2} use a hybrid CNN-LSTM model for personality prediction. They pre-processed social media text data and fed that to an LSTM model, CNN model, and a hybrid CNN-LSTM model. They concluded that the hybrid CNN-LSTM had about 12\% better accuracy than their CNN model and about 8\% better accuracy than their LSTM model at the end of their training.

**Predicting Personality Traits with Semantic Structures and LSTM-based Neural Networks**

M. Kosan et. al. \cite{carl\_3} use an LSTM-based neural network to predict personality traits based on social media data. They use a plethora of steps to preprocess their data, including removing excess characters, making all the text lowercase, translating foreign languages, correcting spelling mistakes, word vectorization, part-of-speech tagging and more. Using LSTM and Bi-LSTM models with swish activation function, adam optimizer, and Mean Squared Error loss, they concluded that the Bi-LSTM model performed better than the LSTM model, and that their own pre-processing methodology performed better than models FastText.

**A Plausible RNN-LSTM-based Profession Recommendation System by Predicting Human Personality Types on Social Media Forums**

V.V.R. Maheswara Rao et. al. \cite{carl\_4} compare an RNN-LSTM-based model against other machine learning algorithms in the task of classifying a person's personality using the Myers-Brigg Type Indicator (MBTI) to recommend a profession. Their model uses RNNs to take context and dependencies in the data into account, and LSTMs to avoid vanishing gradients. The personality classification was broken into four binary classification tasks for each of the components of an MBTI personality class, with a final 97.81\% accuracy on MBTI test data and performed better than KNN, Logistic Regression, Naive Bayes, and SVM.

**Computation Analysis for Identifying the Protagonist and Antagonist and their Sentiments in Harry Potter Books**

D. K. Tayal et. al. \cite{sandhi\_1} used sentence level Valence Aware Dictionary and Sentiment Reasoner (VADER) sentiment analysis to determine the protagonist, antagonist, and neutral character in the seven Harry Potter books. After pre-processing their data, they concluded that Harry Potter has the most positive polarity, Voldemort has the most negative polarity, and Hermione Granger has the most neutral polarity, making them the protagonist, antagonist, and neutral character respectively.

**The Importance of Context for Sentiment Analysis in Dialogues**

I. Carvalho et. al. \cite{sandhi\_2} look into the importance of context when conducting sentiment analysis on conversational style data, specifically Portuguese customer-support conversations. They found that providing context caused models to perform equally or better, with one model improving its score by 29\% when going from no context to two sentences of context. They also concluded that classification will improve if models are provided with more information.

**Implementation of Burmese Language News Classification System by Using SVM and LSTM Machine Learning Algorithm**

K. Z. Ye et. al. \cite{sandhi\_3} classify Myanmar local and international news in social media using Term Frequency Inverse Document Frequency (TF-IDF) feature extraction techniques and compare the results with a SVM ML algorithm and a LSTM RNN. The data was collected, then tokenized and pre-processed to remove stop words and unusable characters. They concluded that the LSTM model is slightly more accurate than the SVM model, but both performed slightly below expectation in real time testing since some of the categories were similar to each other.

**Sentiment Analysis Using Product Review Data**

Fang, X. and Zhan, J. \cite{joaquin\_1} looked at sentiment analysis of Amazon reviews using non-neural network methods since it is dated for 2015 and neural network based modeling wasn't as popular as it is now. They converted the review into tokens that were used as training data for the three models tested: Naïve Bayesian classifier, Random Forest, and SVM. In the end, their best model was able to get an accuracy of 85 percent.

**VADER: A Parsimonious Rule-based Model for Sentiment Analysis of Social Media Text**

C. Hutto and E. Gilbert \cite{joaquin\_2} presenting their own proposed Valence Aware Dictionary for Sentiment Reasoning, or VADER model to sort phrases as positive, negative, and neutral. Their model is particularly equipped to analyze slang such as 'LOL', and emoticons by linking them to the words they mean. It is also simpler and less of a black box compared to other ML models. The results of their experiments was a list of rules that help identify sentiments which include: Punctuation, Capital Letters, Degree modifiers, Contrastive conjunctions (e.g. 'but', 'however', \& Negated sentences

**Convolutional, Long Short-Term Memory, Fully Connected Deep Neural Networks**

T. N. Sainath et. al. \cite{joaquin\_3} describe three popular separate models: DNNs, CNNs, and LSTMs, before describing an experiment about combining the three in different arrangements, architectures, and orders. The authors specifically promote a model they call CLDNN, which combines the strength of all three models together as they believe a CNN can help an LSTM by removing randomness, noise, and frequency shifts and a DNN is the best layer/model for presenting the result of a test. They found that their own CLDNN model scored the highest on the Word-Error-Rate, having a 5\% improvement over the vanilla LSTM model.

The Dataset:  
The dataset we will be using to train our model to sort characters into different houses is the Harry Potter Movies dataset on Kaggle \cite{dataset}, filled with movie dialogue, characters, and other interesting information in a set of CSV files. On the Kaggle website, others have used this dataset to solve similar problems involving language, showing that the dataset is trusted to be an accurate representation of the fictional characters, and therefore the underlying connections between the members of each Hogwarts house. Unlike with the novel version of the book, we will have an easier time identifying who said which line, since it's already marked down in the training data.

We pre-processed the dialogue by:

1. removing columns with irrelevant data
2. removing dialogue spoken by characters who were not in a house
3. removing punctuation
4. removing numbers
5. removing special characters
6. removing excess spaces
7. removing stopwords that carry very little meaning (using stopwords in the nltk library)
8. making all dialogue lowercase

Supervised labels will be based on cross-referencing who said the line in the script, and what Hogwarts house that character is in, which is available in the same data set under a different file. We combined the pre-processed dialogue with the character who said the line and the house they are in. The datasets were then tokenized using tensorflow's Tokenizer, and each vector was 0 padded to match the length of the longest sentence. This formed the input to our model.

Model Description:

The proposed model for this problem is the stacked LSTM, which takes the time-series analysis from the regular LSTM layer, and adds in multi-layer abstraction through the addition of a second layer. The idea is that the first layer would take raw input and identify key features, which are then passed into another LSTM layer which does the time-series analysis. As shown in Fig. \ref{fig}, we are including a fully-connected layer at the end to help consolidate the information from the LSTM layers before going into the output layer, which will be one of the four houses that the character is from.

The pre-processed data was then split into training, validation, and test sets using sklearn train\\_test\\_split with a test size of 0.3 (i.e. 49\% training, 21\% validation, and 30\% testing).

Hyperparamters:

The main hyperparameters used in the this architecture is the number of LSTM modules in each layer. LSTM modules can't be trained using Matrix Multiplication, which means that GPUs are ineffective in improving training speed. Deciding the number of modules to use was a balance between increasing the model's ability to learn and classify, and reducing the amount of training time required. In the end, we selected 40 modules for each of our two layers, which led to a training time of almost a minute per epoch. For the other hyperparameters, such as learning rate and decay, we used the default values as provided by Tensorflow.

Regular Dataset:

Three models with the same architecture were trained on the different iterations of our dataset as described earlier. The first dataset was the original one with no changes made to it.

Shortened Dataset:  
Next, the dialogue was separated into different houses and the number of lines per house was counted. The second dataset was formed by reducing the number of lines for the houses who had more than the house with the least number of lines. The dialogue was shuffled before removing the extra lines to ensure all the included lines were not just from one speaker.

Duplicate Dataset:  
The final dataset was created by increasing the number of lines for the houses that had fewer lines than the house with the most number of lines (Gryffindor). We found a PDF version of the scripts for a spinoff of the Harry Potter series called Fantastic Beasts and Where to Find Them \cite{fantastic\_beasts\_1} \cite{fantastic\_beasts\_2} \cite{fantastic\_beasts\_3} where the main character is a Hufflepuff. We extracted some of his lines from the script and added those to the Hufflepuff dialogues after pre-processing them. We then duplicated the existing lines for each house until they all matched the number of lines Gryffindor had. The duplicate dataset finished training with higher training and validation accuracies than the previous models.

Testing Results:

The Testing split of our Harry Potter Dataset was fed into our trained models to see how accurately they would predict houses. The results of these tests was put into confusion matrices which show the True and Predicted Houses over the testing data.

As seen in the confusion matrix for the Regular model (Fig. \ref{Regular Matrix}), it performed poorly in testing. The model predicted Gryffindor most of the time, which is likely due to the larger amount of Gryffindor lines in this dataset.

The confusion matrix for the Shortened model (Fig. \ref{Shortened Matrix}) shows that model predicted mostly Gryffindor with some Ravenclaw predictions. Similar to the Regular Model, the large amount of Gryffindor predictions is likely due to the fact that a majority of important lines are said by Gryffindors, which can influence the model. In regards to the small bias to Ravenclaw, they are categorized as the "smart ones", which may influence the model to classify any lines as such.

We observe much better performance during testing for the Duplicate model, as seen in its confusion matrix (Fig. \ref{Duplicate Matrix}), which shows that the model mostly predicted the correct house.

Star Wars Testing:

After training a model using the Duplicate dataset, we tested it by sorting characters from the Star Wars movie series. We used the Star Wars Movie Scripts dataset on Kaggle \cite{starwars} which includes the scripts of the first 3 movies (i.e. Movies number IV, V, VI) formatted by line with the character tagged. The data from the 3 movies was concatenated into one dataset and put through the same pre-processing as the Harry Potter dataset.

The characters with the most lines in the dataset were identified as Luke, Han, Threepio, Leia, and Vader. Each of these characters' lines were exported to CSV files and then fed into the model trained on the Duplicate dataset to get predictions.

Luke, Han, Threepio, and Leia are all protagonists, and were classified as Gryffindor by our model (see Fig. \ref{luke} as an example). Vader, a villain, on the other hand, was classified as Slytherin (see Fig. \ref{vader}). These results make sense based on the fact that Harry Potter (the protagonist) is a Gryffindor, and Voldemort (the villain in the Harry Potter movies) is a Slytherin.

Conclusion:

Through training Stacked-LSTM models on Harry Potter dialogue, we were able to sort characters into their respective houses based on their lines. Our model was trained on a dataset with a relatively equal amount of data for each Hogwarts House through duplication which yielded good results when predicting what house a Harry Potter character is in. When tested using characters from the Star Wars movies, results were also as expected. While our testing yielded positive results, further testing with a more diverse range of characters is still necessary to validate the efficacy of our model.

Specific Script things:

Three models with the same architecture were trained on the different iterations of our dataset as described earlier. The first dataset was the original one with no changes made to it. It yielded high training and validation accuracy as seen in the leftmost figure, but that of course was because there were so many Gryffindor examples and it simply picked Gryffindor most of the time.

Next, the dialogue was separated into different houses and the number of lines per house was counted. The second dataset was formed by reducing the number of lines for the houses who had more than the house with the least number of lines. The dialogue was shuffled before removing the extra lines to ensure all the included lines were not just from one speaker. The shortened dataset yielded lower training and validation accuracies than the regular dataset as seen in the middle figure. This is possibly because there were not enough examples to actually train the model.

The final dataset was created by increasing the number of lines for the houses that had fewer lines than the house with the most number of lines (Gryffindor). We duplicated the existing lines for each house until they all matched the number of lines Gryffindor had. The duplicate dataset finished training with higher training and validation accuracies than the previous models as seen in the right most figure.