# Naturalizing the Dungeons and Dragons Experience via NLP

Zach Schickler
zachschickler@knights.ucf.edu

Blake Wyatt
blakewyatt@knights.ucf.edu

## ABSTRACT

Dungeons and Dragons (D&D) is a roleplaying game where several people engage in conversation to progress the game in the form of an adventure. At it's core, it is a pen and paper game that requires many books, papers, dice, and figurines [1]. Moving D&D to an online setting has several benefits, including a lower entry price, convenience, and access to a wider base of players. But, moving online is held back by immersion. We propose a model that improves the flow of game in online D&D by assisting the Dungeon Master (DM) in selecting the right action when a request by a player is made. We train the model on data derived from transcripts of Critical Role gameplay [2] [3]. We find that a model of this nature is possible and that a full tool with Discord integration is feasible as a means of making online D&D easier.

## 1. INTRODUCTION

Dungeons and Dragons is a roleplaying game in which a group of people get together and essentially engage in a conversation to play a game. This conversation is not entirely freeform; it follows certain rules that dictate how people interact. There are several players and one Dungeon Master. The DM is in charge of playing everything that the players interact with. Whenever a player comes across a non-playable character (NPC), the DM must play as the NPC. Additionally, the DM manages the state of the game and is the players' "bridge" into the world. Players will communicate with the DM in specific ways. A player may tell the DM where they intend to move within the fantasy world. A player could also ask questions about what is happening inside of the world, and then the DM decides what the player must do the acquire this information. While D&D is a very freeform game with rules differing from DM to DM, we decided to base our classification off of the most general standards we could come up with, which are based on our own game experience and that of Critical Role.

Critical Role is understood to be the name of a famous group of voice actors that play and stream D&D to fans. They operate under the Critical Role Foundation, a non-profit organization that holds charity fundraisers while streaming D&D [2]. Critical Role is possibly the most famous example of D&D being played for an audience. Their games are also available as podcasts on several different platforms. We look specifically at their first campaign, *Vox Machina* [2]. *Vox Machina* contains a total of 115 episodes, each spanning 2-3 hours long. Entire transcripts in the form of JSON files for each episode in *Vox Machina* can be found on Github, provided by Rameshkumar et al. [3].

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

## 2. RELATED WORK

### 2.1 Text Classification

Text classification is a Natural Language Processing task with a multitude of algorithms and techniques surrounding it. Learning from modern techniques to improve text classification is beneficial to developing an effective way to perform our specific task.

Text preprocessing is an important component of text classification. Text preprocessing involves filtering through initial data to remove unnecessary components, and at times formatting the data in a way that makes it more useful for the task at hand [4]. Text preprocessing can involve removing filler words that do not contribute towards the unique identification of and classification of sentences. In our specific task, we will identify several key word phrases that carry the majority of the significance in terms of classification related to the task.

Tokenization is another concept in text classification that should be considered. Sentences will often be split up into a format where each word is uniquely recognized. In text classification, it can be useful to allow tokens to share some sort of interconnectedness due to their mutual dependence. For example, "head to" would be regarded as two tokens, "head" and "to". Yet, recognizing their connection can help in reaching higher accuracy in text classification.

### 2.2 Opinion Classification

One such example of a text classification model that is worth studying is in a paper by Othman et al. [5]. This model classifies opinions by types as outlined by the paper. It serves a purpose very similar to our model because it classifies sentences by their semantic derivation. Their model picks up on key words that are qualifiers, such as "very" or "hardly", to determine the extremity of the sentence before giving proper classification.

### 2.2.1 Tagging

The paper by Othman et al. utilizes a technique called Tagging. Tagging occurs in a preprocessing stage where the input is tokenized, and then key words are tagged before being presented to the model. In the model, tagged words carry a modified weight that more significantly impacts the results of the model. This technique greatly speeds up the learning process for a model by guiding it in the right direction in terms of what certain words mean. This problem is very similar to our problem, since D&D is a game guided by certain action words. For example, movement actions are one such possible classification that can arise from our input. If words like "head to" and "go" are always seen in the input for movement actions (which they are), then we can tag these words to tell the model that it is almost a guarantee that these sentences will be classified as movement actions.

# 3. METHODOLOGY

## 3.1 Data Collection

The proposed model requires data that is very specific, and is as such very hard to collect. We were very limited on what data we could use, but decided on a dataset compiled and preprocessed by Rameshkumar and Bailey [3]. This paper contains, among other things, full transcripts for the entirety of the first campaign of D&D played by Critical Role, *Vox Machina*. With 115 episodes and close to 300 hours of dialogue, sorting through this data is an extremely time intensive task. The primary issue that was encountered with this data is that it is currently in a form that is unusable to us. The dialogue is provided as JSON files, where each series of utterances is tagged with a speaker. Matt is the name of the DM, so any utterances tagged with "Matt" are spoken words by the DM. For our experiments, we are only interested in the words that are spoken to the DM, so we can use this information to narrow down what data we need to look at when refining data. From this point, we only want initial utterances to the DM.

### 3.1.1 Classes Design

We decided on four classes that these "action phrases" can fall under. The first classification is combat. Combat action phrases indicate that a player is trying to request to the DM that they want to use an ability. This could be a spell or any action that would be considered a combat or out of combat action. One such example could be "I'm going to shoot a Fireball" [3]. In this action phrase, the player is requesting to the DM that they use an action, which would be followed by the DM taking the necessary actions to let this action occur.

The next classification is movement. Movement actions are phrases where a player requests to change their position to the DM, whether it be just them moving or a movement suggestion to the entire party or parts of the party. One such example is "I'm going to saunter on up to the door" [3].

The next classification is non-combat actions. This classification is a bit more complicated than the previous two actions. Any phrase considered a non-combat action must be a phrase that does not attempt to engage in combat, move, or interact with an NPC. These phrases can be considered meta-phrases in the sense that they are spoken often out of character (OOC) directly to the DM. While speaking OOC is often an indicator that the phrase is a non-combat action, it does not dictate this entirely. Non-combat actions make up the majority of actions in D&D, and are primarily used to gain information about the fantasy world. The DM can sometimes choose to make players roll a dice to decide the quality of the information they receive, but this dice roll is at the complete discretion of the DM. Initially, we were going to split dice rolled non-combat actions and non-dice rolled non-combat actions into two separate classifications. We decided against this practice due to the ambiguity and lack of distinction between the two. We ultimately decided to merge them into one classification to improve the results of the model.

The fourth and final classification is when a player is talking to an NPC. Talking to NPCs is an important part of the roleplaying aspect of D&D. As previously mentioned, the DM is responsible for playing as any NPC at any given time. Characteristics for this classification are more broad than the other classes. Whereas movement and combat have very specific phrases that indicate its classification, talking to NPCs is more closely aligned with natural conversation. One such giveaway is the increased use of proper nouns and fantasy-esque words. While these indicators are more fluid than others, they are still useful.

### 3.1.2 Data Formation

Data was compiled through a process in which we manually scanned the Github-hosted JSON files for action phrases, and then formatted these action phrases in a Google Sheets spreadsheet, along with our assigned classification for what we understood the action phrase to be based on the context of the episode. This process required a moderately good understanding of how the game D&D is played.

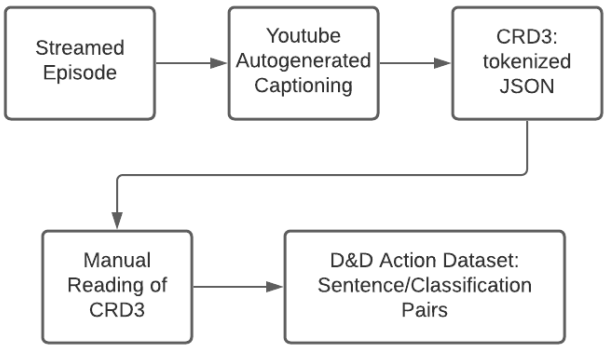


Figure 1: A pipeline showing the full formation of training data is shown.

In total, 200 rows of data were collected from CRD3, forming the D&D Action Dataset. This process took several days. We will later discuss the repercussions of our dataset size. During experimentation, this data was split 70%/30% for training and validation.

## 3.2 Text Classification Models

One major component of the project was selecting the best model to use for experimentation and transfer learning. Because the dataset being used is very small, it would be impossible to use the data to train from scratch a model

that could achieve any level of success. The next logical step when faced with this problem is to look for a model that is pre-trained with pre-existing weights. From this point, we can simply modify the output layer of the model to fit our classification. There is no need to change the input, since we are still just providing spoken English sentences. We explored several models which all fall under the categorization of transformer networks.

Transformer networks utilize a principle called Attention [6]. During the attention phase, more computing power and emphasis is put on a smaller portion of the input, whereas less resources are being spent for the greater portion that is considered not as important. This technique lets the model remember key features about an input, helping it focus on and catch repeated meaningful portions [7]. In our example, this would mean catching key phrases mentioned earlier, such as "cast" and "head to", while ignoring flavorful gibberish that D&D games often have in the form of roleplaying personalities.

We decided to use Xlnet, a transformer network published by Yang et al. [8]. This network showed improved results over its state-of-the-art competitors, and had extensive documentation making working with the model easier.

### 3.2.1 BERT

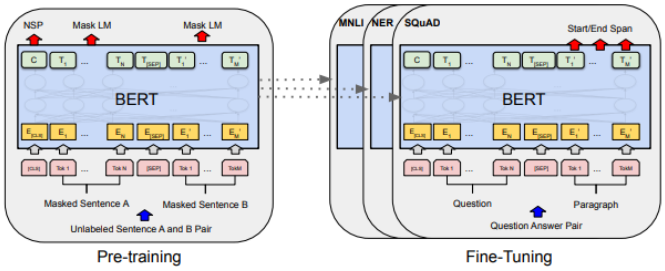


Figure 2: The BERT architecture [9].

One model that was analyzed is Bidirectional Encoder Representations from Transformers (BERT), from Devlin et al. [9]. The BERT model is the precursor to the other models that will be mentioned, and uses a strategy of pretraining and fine-tuning to create a powerful text classification model that can be modified for any desired output layer. We did not choose this model because there are several improved options in terms of performance and accuracy.

### 3.2.2 Xlnet

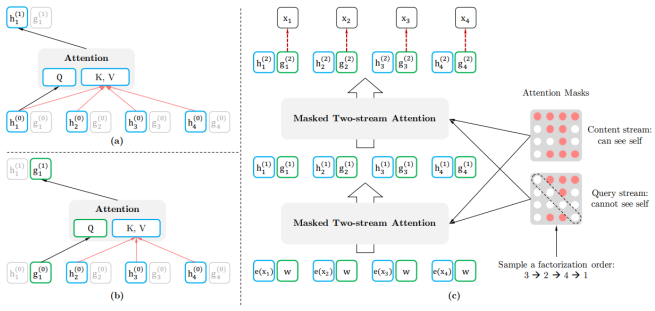


Figure 3: The Xlnet architecture [8].

Xlnet is a direct improvement to BERT, aiming to fix the major problem that BERT has. BERT suffers from a pretrain-finetune dependency [8], which Xlnet solves through the use of a process called autoregressive formulation [8]. Xlnet had the best performance and documentation out of all of the models, which is why we decided to perform our experiments with Xlnet.

### 3.2.3 XLM

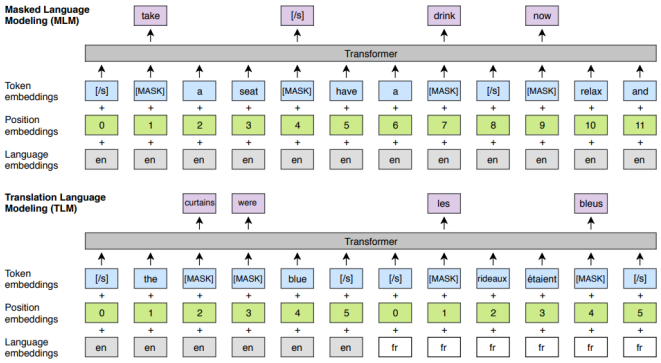


Figure 4: The XLM architecture [10].

XLM makes use of a technique called generative pretraining [10]. This technique is supposed to be especially effective for English NLP tasks. While XLM is a good candidate, we decided to use Xlnet over documentation concerns.

### 3.2.4 RoBERTa

RoBERTa solves two major issues from BERT. The first issue is that Liu et al. [11] believed that BERT was severely undertrained, which led to accuracy drops. The second issue is that they believed BERT did not have the best chosen hyperparameters. RoBERTa managed to find accuracy improvements over BERT and be a valid contender against other models.

### 3.2.5 DistilBERT

DistilBERT does something similar to RoBERTa by being a direct imrprovement on BERT. However, DistilBERT instead aims to make BERT more lightweight and easier to run on low end machines [12]. DistilBERT achieves these goals, but does not see significant improvements over just using Xlnet.

## 3.3 XLNet Model Adjustments

After we determined XLNet to be the text classification model of choice, we researched how to best implement it. We found there were multiple implementations we could leverage. One was using XLNet's paper's original source code, another was using an implementation found on Github, and another was using the 'transformers' Python library. There were other options, however, these three different implementations seemed to be the most likely to be true to the paper due to their wide use.

Ultimately, we settled on using the transformers Python library. The transformers library offered multiple implementations of XLNet to suit whichever case we may need and many helpful tools to handle the minutia of XLNet like a

Tokenizer to parse strings and convert them into the format XLNet needs to train and classify. Additionally, the transformer library is used by a large number of developers, there is a large amount of support for its implementation, a minimal amount of bugs, and relative easy-of-use.

To implement the XLNet model for our text classification problem, we made use of multiple resources given in the transformers Python library. The first most notably and most critical, is the pre-trained XLNet text classification model. Defined in the library as 'XLNetForSequenceClassification' and 'xlnet-base-cased', we obtained the pre-trained version and replaced the final layer of the pre-trained XLNet model with a new fully connected classification layer of four nodes corresponding to the four action labels given in our DD Action Dataset. However, XLNet relies on a unique form of input when passing data to the model. For example, one cannot simply pass a string as an argument to the model and expect it to train. There are other arguments such as the input ids and attention masks. If we recall, XLNet uses attention to achieve greater accuracy and that comes with the additional complexity of attention masks. Additionally, the text must be formatted and parsed in a specific way. Luckily, the transformers Python library is capable of making this simple and handling the string parsing complexity for us. This is done with the 'XLNetTokenizer' string tokenizer. We specify the exact XLNet model for which to tokenize, obtain the tokenizer, then apply it to all of our text. The tokenizer parses and outputs the sentences as input ids which we can then pass as input to the XLNet model. Our DD Action Dataset is formatted as a CSV file with two columns: a sentence column and a label column. Each row of the CSV is a datapoint and the sentence and label correspond to each other. We read this file into Python, tokenize it with the given tokenizer, and feed it to the model for training or validation as needed.

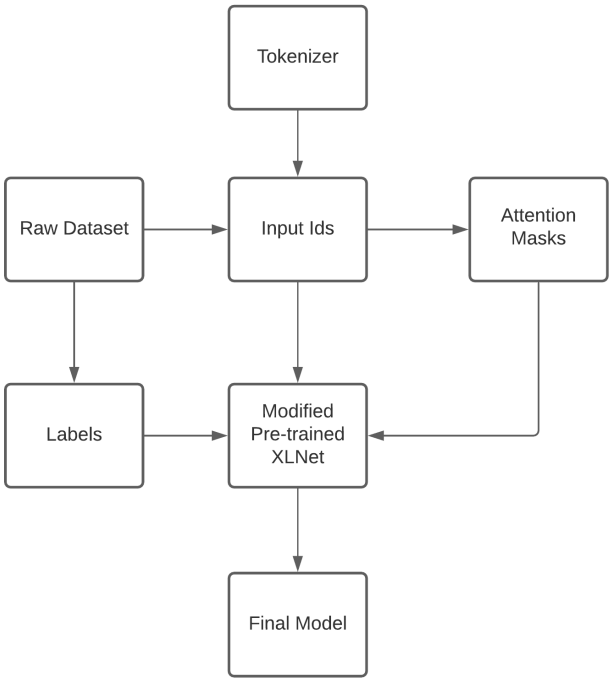


Figure 5: A pipeline demonstrating the code implementation of Xlnet for experimentation using our output layer.

| run | batch_size | epochs | layers_frozen | learning_rate |
|---|---|---|---|---|
| 0 | 32 | 4 | false | 2.00E-05 |
| 1 | 32 | 4 | true | 2.00E-03 |
| 2 | 16 | 4 | false | 2.00E-05 |
| 3 | 16 | 16 | false | 2.00E-05 |
| 4 | 16 | 16 | false | 2.00E-05 |
| 5 | 16 | 64 | true | 2.00E-05 |
| 6 | 16 | 64 | true | 2.00E-07 |
| 7 | 8 | 16 | false | 2.00E-05 |

Table 1: The configurations for each run using Xlnet are shown.

| run | train_loss | train_accuracy | validation_accuracy |
|---|---|---|---|
| 0 | 1.238 | 44.063% | 0.350 |
| 1 | 1.438 | 25.521% | 0.200 |
| 2 | 1.098 | 56.250% | 0.500 |
| 3 | 0.045 | 99.479% | 0.813 |
| 4 | 0.046 | 99.917% | 0.766 |
| 5 | 1.427 | 34.028% | 0.281 |
| 6 | 1.381 | 31.713% | 0.156 |
| 7 | 0.006 | 100.000% | 0.797 |

Table 2: The results of experimentation using Xlnet are shown.

## 4. EVALUATION

To train our XLNet classification model, we varied hyperparameters while training on the pre-trained XLNet model with a change in the final fully connected classification layer. These changes in hyperparameters include the batch size, number of epochs, whether or not the pretrained layers were frozen or not, and the learning rate. Afterwards we evaluate it using the training accuracy and validation accuracy. We vary batch size from 8 to 32, epochs from 8 to 64, pre-trained layers frozen from true to false, and the learning rate from 2e-3 to 2e-7.

## 5. DISCUSSION

Conventionally, when fine tuning on a pre-trained model for another classification task, you replace the final classi-

fication layer with a new classification layer corresponding to the new classification task, freeze the weights in all pre-trained layers up to the final, new classification layer, and then train that final layer on the data for the new classification task. However, in our case, this proved to be insufficient. As shown in Tables 1 and 2, when training with on frozen pre-trained layers, validation accuracy does not exceed 28%. However, when training on unfrozen pre-trained layers, we are able to achieve a validation accuracy of 81%. We hypothesize the reason for this is do the the size of the DD Action Dataset. It is relatively small and therefore is not able to generalize well over all the use cases the pre-trained model is accustomed to. When we unfreeze the pre-trained layers, we are then able to adjust the pre-trained model itself to make it more accustomed to our data. Unfortunately this hurts generalization, but when one is limited by data, you try to achieve the best accuracy you can.

We varied many hyper parameters when training our text classification model to determine what works best. We found that on this small dataset, epochs greater than 16 seem to have minimal, if no effect on validation accuracy. As one reduces the amount of epochs below 16, the validation accuracy often decreases when training with unfrozen layers. Training accuracy also often reaches 100 or extremely close to it. This is an indicator we are greatly overfitting to our dataset and can be proven with the validation accuracy which is not nearly as accurate. To avoid this, conventionally one trains for less epochs and increases the size of their dataset if possible. We varied the batch size while training, however, we were limited to less than 32 while training on our GPU. A larger batch size resulted in a memory capacity error. Despite that we did run a few experiments using the CPU to overcome that limitation. We can see in Tables 1 and 2 that as the batch size decreases from 32 to 16, we achieve a greater validation accuracy. As the batch size decreases from 16 to 8, we achieve a similar validation accuracy as with 16, however, there isn't much data for this batch size, therefore, it is unclear whether this is truly the case and not a result of chance. Since increasing epochs to 64, did not change much over those additional epochs, we attempted to leverage the benefits of a smaller learning rate to more slowly reach our best accuracy and hopefully reach an even greater accuracy. However, this only proved to do worse in both cases attempted.

We ran into several GPU related issues when doing the experiments. These issues resulted from version mismatches that resulted in not detecting a GPU. These issues were resolved by resorting the experimentation on CPUs. Running on CPUs resulted in a noticable slowdown, which meant we had to cut back the number of epochs

# 6. CONCLUSION

We determined that some level of success can be achieved in a model with our intended goal. The model was successful at classifying our inputs, which can be useful in a way that assists the playing of D&D. It is possible that some overfitting of the data was seen as a consequence of using too small of a dataset. This was unforeseen by us since we did not anticipate the challenge that comes with manual collection of data of this nature. One possible improvement of this experiment in the future would be to use more data. We used a dataset of 200 entries, split 70%/30% for training and validation. For repeats of this experiment, we would recommend using a dataset x5 larger, in the range of 1000 entries. We would keep the training and validation split the same. On higher epoch runs, in the range of 64, we did notice slight overfitting. This was not as noticeable on runs whose epochs were in the range of 4-16. On runs with smaller epochs, we believe that key words were in fact being picked up by the model.

One observation that was made about the data when collecting it is that, despite the freeform structure of D&D and its organic playstyle, action phrases were more or less always following the same format. What is more interesting is that there are never any official guidelines for how action phrases must be uttered when playing the game. This leads us to believe that there is an unspoken but commonly accepted meta for how to play the game that develops purely from repeated playing. In other words, the community has largely defined how different actions must be spoken. We think that this could be an interesting phenomenon to investigate in future research.

Xlnet certainly contributed towards our success in this experiment. Without a pretrained transformer network, we would not have had any success due to the very limited size of our dataset. While experimenting with other models would be an interesting comparison to make, it was not in the scope of this project. A possible next step in this research would be to experiment with all of the candidate models mentioned on similar setups, and compare their results. Our hypothesis would be that Xlnet would perform the best for the reasons described in the paper.

# 7. WORK DISTRIBUTION

## 7.1 Zach

Zach was responsible for finding the data to be used in the experiment. Additionally, Zach compiled the new D&D Action Dataset manually. Zach designed the classifications to be used with the data. Zach did research into transformer networks and analyzed several state-of-the-art transformer networks for use. Zach outlined the reasons for using Xlnet, as well as explaining why it beat out other networks. Zach described, outlined, and visualized the problem including the problem statement. Zach led development on the presentation of this research paper.

## 7.2 Blake

Blake implemented the transfer learning text classification used in the experiment. Blake led the development on adapting the model to be used for our output. Blake ran the experiments on his machine to get the results used in the paper. Blake also spent time debugging issues with the model to make everything run smoothly. Blake assisted in the development of presentation of ideas in the presentation.

# 8. REFERENCES


[1] "What is d&d? | dungeons & dragons."

[2] "Critical role."

[3] R. Rameshkumar and P. Bailey, "Storytelling with dialogue: A critical role dungeons and dragons dataset," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 5121–5134, 2020.

[4] K. Kowsari, K. Jafari Meimandi, M. Heidarysafa, S. Mendu, L. Barnes, and D. Brown, "Text classification algorithms: A survey," *Information*, vol. 10, no. 4, p. 150, 2019.

[5] M. Othman, H. Hassan, R. Moawad, and A. M. Idrees, "Using nlp approach for opinion types classifier," 2015.

[6] "Transformer (machine learning model) - wikipedia."

[7] "Attention (machine learning) - wikipedia."

[8] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. Salakhutdinov, and Q. V. Le, "Xlnet: Generalized autoregressive pretraining for language understanding," *arXiv preprint arXiv:1906.08237*, 2019.

[9] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[10] G. Lample and A. Conneau, "Cross-lingual language model pretraining," *arXiv preprint arXiv:1901.07291*, 2019.

[11] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," *arXiv preprint arXiv:1907.11692*, 2019.

[12] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter," *arXiv preprint arXiv:1910.01108*, 2019.

[13] "Discord | your place to talk and hang out."

[14] "Github - revanthrameshkumar/crd3: The repo containing the critical role dungeons and dragons dataset.."