IMPLEMENTING ARTIFICIAL INTELLIGENCE INTO TEAM DECISION MAKING

by

SHARDUL DESHMUKH

(Under the Direction of Neal Outland)

ABSTRACT

The problem aimed at is the suboptimality of team decision-making. This thesis performs an attempt at implementing an artificial agent into a conversation to facilitate team decision-making. Two methods are experimented with to attempt to achieve this goal. The agent records and analyzes under and over speaking within participants of the team. The agent also tracks conversation topics and generates text to add relevant information into the conversation based on what the team is currently discussing. The thesis refers to the first method as ‘moderating speaking equity’ and to the second method as ‘inserting conversation relevant information’. Moderating speaking equity is performed through a python script and inserting conversation relevant information is performed through two separate Recurrent Neural Networks (RNNs) with Long Short-Term Memory (LSTM) layers and Generative Pretrained Transformer J (GPTJ).

INDEX WORDS: [Team Decision Making, Recurrent Neural Networks, RNN, Long Short-Term Memory, LSTM, GPTJ]

IMPLEMENTING ARTIFICAL INTELLIGENCE INTO TEAM DECISION MAKING

by

SHARDUL DESHMUKH

BA, University of Georgia, 2021

A Thesis Submitted to the Graduate Faculty of The University of Georgia in Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE

ATHENS, GEORGIA

2022

IMPLEMENTING ARTIFICIAL INTELLIGENCE INTO TEAM DECISION MAKING

by

SHARDUL DESHMUKH

Major Professor:	Neal Outland
Committee:	Frederick Maier
	Drew Abney

Electronic Version Approved:

Ron Walcott
Vice Provost for Graduate Education and Dean of the Graduate School
The University of Georgia
May 2022

DEDICATION

I dedicate this to my family, who has supported me through all my endeavors.

ACKNOWLEDGEMENTS

I would like to acknowledge my major professor, Dr. Neal Outland, as well as my committee members, Dr. Fred Maier, and Dr. Drew Abney. They have provided me with valuable support and guidance which I could not have gone without. I would also like to further emphasize my appreciation for the time and wisdom that Dr. Neal Outland has provided me throughout this entire process. I would also like to acknowledge the research assistants, Caleb, Canada, and Heidi who have helped me evaluate the results of this study. All these contributions are greatly appreciated.

TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

CHAPTER 1

INTRODUCTION

Team Decision Making, as it has been consistently studied, is often suboptimal (Brodbeck, 2007). Instead of there being an even spread of information among all team members, there is usually a single person leading the team discussion (Bales, 1951). This could lead to a myriad of problems when it comes to arriving at an optimal solution. Since one or few people are influencing most of the information presented, it is often found that the opinions and information that the dominant person offers is what is most heavily considered (Chahine 2017). This exacerbates the problem of uneven information spread and could lead to a suboptimal decision being made. The issue of suboptimal decision making in team decisions has impacts everywhere from the classroom to global politics. Decisions about class projects to military strategy are made in teams, and if team decision making is found to be suboptimal, then nearly all decisions made in teams may be under question as well.

A series of best practices could be utilized to alleviate these issues. Some best practices include setting clear goals, redefining tasks, holding second-chance meetings, developing new norms, and encouraging minority influence. Most of these best practices are accomplishable through training team members. Some best practices, however, are more analytical and more easily convertible to a computational task. Artificial Intelligence may be able to efficiently address these issues.

Computer-based agents have already been experimented with to moderate participation rates between team members (Kim, 2020) and automated agents such as Apple's Siri are being

used to execute tasks such as setting reminders and relaying answers to questions (Canbek, 2016). AI is thus well positioned to influence minority influence and dissent in teams. To do so in a team decision making setting, AI must be able to understand human interaction and be able to communicate with team members. However, teams may be less likely to be influenced by outside members (Groom & Nass, 2009). Thus, for AI to be successful in ameliorating ills in team decision-making, it may be best to incorporate AI capable of serving team functions, thus aiding the team in accomplishing task goals.

This thesis attempts to insert an artificial agent in a task-oriented group conversation and evaluate whether the agent can moderate the speaking frequency of the members to ensure the flow of information is as equitable as possible. This tenet of the research will be referred to this as the agent *moderating speaking equity*. This functionality of the artificial agent will attempt to equalize the distribution of participation amongst members. In other words, it will try to limit the speaking duration of someone who may be dominating the conversation, and it will try to promote the participation of someone who is less active in a conversation. This could hopefully promote the spread of all the available information held within the team and possibly lead to the outcome of a better, more informed decision.

An additional aim is to observe if an artificial agent can also add information to the conversation as if it were a team member. This information should be relevant to the group discussion and should ideally also add to the process of the team making an optimal decision. Thus, agents must effectively recognize which topic the team is currently discussing. If the agent were to misread the topic of discussion, it would be impossible for it to add in relevant information about that topic. The agent also must have a relevant store of information about the

topic it detects, so domain specific knowledge must be inserted into the model. Finally, the agent must also be able to generate intelligible text.

For agent communication capabilities, the model will leverage a text processor known as Generative Pre-trained Transformer J (GPTJ). GPTJ, in summary, is a free open-source version of GPT3 and creates text outputs following conversational inputs. There are multiple use cases for this technology. This includes conversational speech generation, which is what this study will be utilizing it for, but also code generation and narrative generation (Matiana, 2021). It is robust enough in text generation that it was used to complete college level homework assignments without being flagged by the market standard plagiarism detection system (Biderman, 2022). GPTJ is built using 6 billion parameters and is one of the modern leaders in creating intelligible text.

The conversations that the proposed agent will be inserting itself into are conversations about the Winter Survival Task. This task is a group activity where the members need to work together to form a consensus about how they will use items to survive in a desolate winter environment. The instructions and details of the task are provided in Appendix A.

1.1 Main Goals of Research

The aim is to facilitate team decision making through two goals: moderating speaking equity and inserting conversation relevant information. These two goals aim to provide a solution to the problem of uneven speaking distribution and advance the conversation further. Moderating speaking equity involves inserting an automated agent into team conversations and assessing ability to administer timely interventions. Inserting conversation relevant information experiments to what degree of accuracy the bot can recognize topics and to which degree of intelligibility the bot can respond with relevant and useful information.

CHAPTER 2

LITERATURE REVIEW

2.1 Research on Teams

In teams research, there is strong evidence that teams have uneven distributions of participation. In small groups of three to ten people, an uneven distribution of speaking is observed among all group sizes (Bales, 1951).

Figure 1 demonstrates a visual representation of this unequal distribution. This appears to work

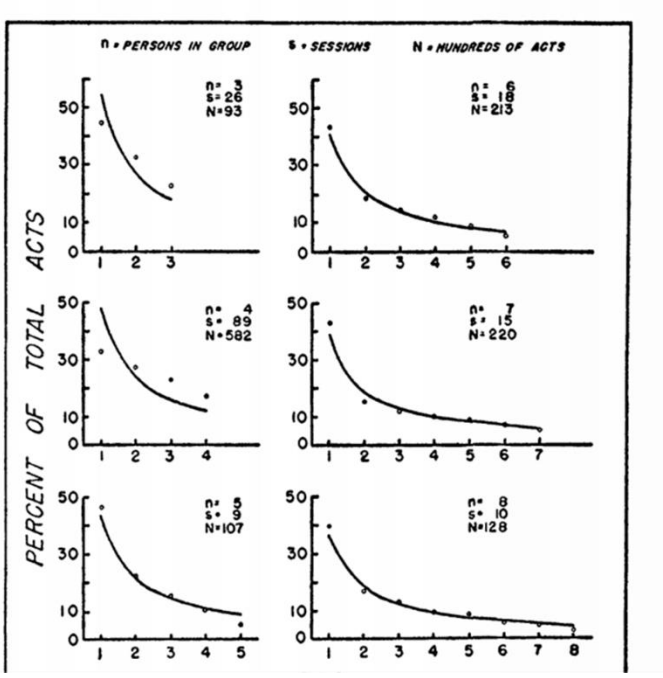


Figure 1: Group Speaking Distributions

on a logarithmic distribution. As shown in the figure, the least active speaker is speaking significantly less than the most active (Bales, 1951). Unevenness in speaking time could also lead to unevenness in the influence of the group's overall decision. Individuals who are more

open and talkative may have a greater influence on the decision-making process than those who are shy or reserved (Chahine 2017).

An asymmetrical distribution of information is what is known as a hidden profile (Stasser, 1988). A hidden profile occurs when the best overall decision is hidden because of differing information the group has not yet shared. The path to arriving at the best overall decision includes sharing all relevant information. If there are individuals in the conversation who are less active, the chances of information going unshared is higher, thus the chance of making a suboptimal decision is higher. Hidden profiles are often the case for teams engaging in decision-making tasks given teams are often composed of individuals diverse in skills and information (Stasser, 1988).

Various attempts have been made to ameliorate the issues in team decision-making. Some known best practices of decision making are (Thompson, 2008):

1. Developing norms that require disagreement
2. Encourage minority influence
3. Define the task as an information-sharing problem, rather than a judgment to be made.
4. Hold second-chance meetings.
5. Set clear goals.

Developing norms that require disagreement include making members aware of potential biases and asking for support in counteracting them (Thompson, 2008). For this to be successfully accomplished, the participants would need to feel a sense of comfort to disagree freely. This is possible if these norms are communicated well and understood by all members of the group. Defining tasks as an information sharing problem is also an interpersonal task that requires the whole group to fundamentally think about the task differently. This best practice

also necessitates good communication between the participants to be accomplished. Holding second-chance meetings is up to the discretion of the person who oversees setting the goals. This is successfully done when the meeting organizer allows for tasks to be revisited in a subsequent meeting. Setting clear goals occurs when the group understands exactly what is needed to be accomplished. These are clear definitions that would need to be communicated by the participants of the group. Encouraging minority influence can successfully occur when less active speakers are identified and prompted to share their input.

It is possible that AI could be implemented to handle each of these best practices, however, each practice would require varying amounts of difficulty to implement. Machines excel at handling numerical data, so the aspects of a conversation that can be easily represented numerically could be most proficiently handled by AI (McCann, 1992). Developing norms that require disagreement would require a large amount of interpersonal communication with the participants in order to convince each member to adopt a new set of norms. Defining the task as an information problem would entail the same set of difficulties. Holding second-chance meetings would require the agent to take a leadership role within the group which is also an interpersonal task. Setting clear goals would be possible, however, a large amount of prior information about the motivations of the group would be required for the algorithm. All these best practices are largely interpersonal tasks while encouraging minority influence is an analytical task, making it a suitable problem for AI to attempt. It requires that the speaking frequencies of the group are calculated, and the appropriate members are prompted based on some algorithmic decision.

2.2 Research on Moderating Speaking Equity

There has been a plethora of studies that experiment with the idea of moderating a conversation using technology (Kiesler, 1984; Stasser, 2006; Kim, 2020; Hogan, 2021). When groups given a task are moderated by computers, they may complete the task a higher solution rate than those groups who were not moderated by a computer (Stasser, 2006). Similarly, when comparing computer mediated groups with face-to-face groups, the computer-mediated groups possibly have more even participation (Keisler, 1984). A participation moderation system implemented in a group messaging context also found that the participation moderation bot helped the group to reach consensus by allowing the members to more effectively identify each other's views (Kim, 2020).

Conversation moderation models have been implemented in various group-decision making tasks. The travel task is a commonly used decision-making task (Kim, 2020; Hong 2018). The task involves participants planning a one-day tour of Korea for a foreign friend (Kim, 2020). The travel task is sufficient for observing how groups interact in a conversation, however, there is no true right answer.

Diplomat is another digital conversation moderation agent (Hogen, 2021). The aspect of Diplomat that is most intriguing regarding moderating speaking equity is the under speaking and over speaking notifications it implements. The design of these notifications is: if a user does not speak for 8 messages in a row, they will be sent an under speaking notification, and conversely if a user is seen speaking for more than half of the last 8 messages, the user will be sent an over speaking notification. This methodology also differs from other studies that calculate under-speaking from number of words instead of amount of time spoken (Kim, 2020). These studies implement these intervention systems because the data they utilize is generated as text-based

data; the conversations are held through a group message (Kim, 2020), Slack (Hogan, 2021), or another text-based software. The participation distribution may be different when transcribing in-person meeting instead. In person transcriptions could allow for interventions based on time spoken rather than number of words spoken. The goal is to expand on models such as Diplomat's under and over speaking notifications by experimenting with differing time windows based on duration of speech on when to intervene into the conversation instead of relying on their model of a certain number of messages.

Diplomat also developed a system for inserting relevant information into the conversation, however, their methodology was not very well received in their results (Hogan, 2021). In their preliminary study, the authors found that the users who were most critical of the agent were those who experienced the feature where it would intervene when there was a lull in conversation. The methodology of this feature is whenever it detects a two-minute silence, the agent sends a link of new topics for the group based on a google search about the initial prompt the group was given (Hogan, 2021). The information given was likely not relevant enough to the current conversation to be beneficial.

2.3 Research on Intent Recognition

Intent recognition is integral for achieving the goal of inserting relevant information into a conversation. Intent recognition is a classification process in which language is classified into the different types of possible 'intents' the programmer decides. These intents can be utilized for classifying customer service requests (Vasquez-Correa, 2021), topics of conversation, or general questions (Goo, 2018).

The use case of intent recognition in question topics can be utilized in detecting intent in a question-and-answer formatted conversation (Indulkar, 2018). The data that the authors

conduct their study on is the Stanford Question and Answering Dataset (SQuAD). The number of possible intents in their original dataset was around 500, but they manually changed this to a much smaller and more manageable total of 18 possible intents (Indulkar, 2018). Then with the data they extracted, the authors used word embedding to put all the words into a word vector in a continuous bag of words model (CBOW). To perform the word embedding they used a Log-Linear Model approach, commonly called as Word2Vec (Indulkar, 2018). They use this as a method of getting the program to calculate word representation in a vector space. For example, the word "project" in their research is represented as a vector of 100 dimensions (Indulkar, 2018). Word2Vec is also context independent since it has a unique vector for each word regardless of its context (Vasquez-Correa, 2021). The Continuous Bag of Words model predicts the probability of a target word from the context in the source sentence (Indulkar, 2018). The CBOW model is also shown to produce accurate results when processing large corpora (Bhoir, 2017). The model structure that typically works well with this implementation is a Convolutional Neural Network with a Long Short-Term Memory layer (Indulkar, 2018; Vasquez-Correa, 2021). The most interesting part of this intent recognition research is the decision to utilize a CNN with an LSTM while other papers elect to use an RNN instead (Lee, 2018). These models do, however, achieve a high accuracy so these models important to consider when deciding upon which model and architecture to use for the research in this paper.

The proposed model will handle free-form conversations about a group's decision-making process. This type of conversation includes long sentences that do not frequently occur in a dataset of question and answering, however, the process for detecting intent is similar. Word2Vec does not model the order of sentences, thus the algorithm is limited to shorter, simpler sentences (Lorenc, 2021). For more complicated tasks, algorithms like a Recurrent

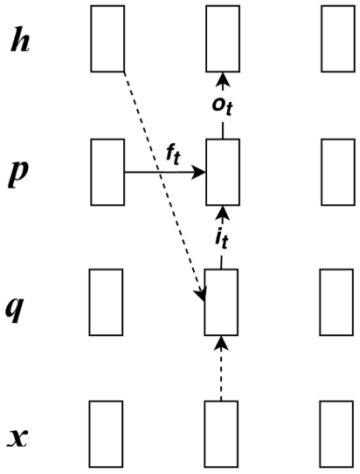
Neural Network (RNN) preserve word order (Lorenc, 2021). The proposed model will be using data that is collected across a span of time and the recognition of intent will be contingent on what is spoken in order. It is important to consider these other methods of intent detection, however, the proposed model will utilize a Recurrent Neural Network (RNN).

Recurrent Neural Networks are better equipped to handle units in sequence (Yin, 2017). Intent recognition using Recurrent Neural Networks can incorporate the same type of LSTM layer which was implemented in the CNN architecture (Hakkani-Tur, 2016; Indulkar, 2018). One possible architecture to implement is a bi-directional RNN with an LSTM layer. This found a 96.4 percent accuracy when identifying intents in a question-answering context (Hakkani-Tur, 2016). The authors of this architecture also experimented with word2vec in their model and found no significant performance improvement. The authors instead opted for one hot word vectors (Hakkani-Tur, 2016). One-hot encoding is a vector that is the size of the vocabulary where the target word is encoded as a 1 and all other words as 0 (Ravuri, 2015).

LSTM can also be implemented for solving the problem of models forgetting conversation context over time (Lee, 2018). The ability to keep context information readily available without forgetting it is essential for keeping up with group conversations. The authors who implemented an LSTM for this use case elected to utilize a RNN model in their research because of the sequential nature of the conversations in their dataset (Lee, 2018). The algorithm they implemented for intent recognition consisted of three RNNs with LSTMs (Lee, 2018). In all three neural networks they used the Adam optimizer and a learning rate of 0.001 which was also used in studies that implemented a CNN architecture (Indulkar, 2018). An LSTM structure is depicted in Figure 2. The architecture of an LSTM includes an input gate i_t, a forget gate f_t, and an output gate o_t. As shown in the figure, the gates are generated by a sigmoid function over the

input state x and the preceding hidden state h. The input state and the hidden layer are combined to produce a temporary result in q which then combines all three input, forget, and output states to produce an updated history in p (Yin, 2017).

Figure 2: LSTM (Yin, 2017)



(c) LSTM

CHAPTER 3

PROPOSED MODEL

Table 1: Goals

Sub-goal	Main Goal	Mechanism
Moderating Speaking Equity	Moderating Speaking Equity	Python Script
Topic Detection	Add information to the conversation as a team member	Intent Recognition
Ranking Detection	Add information to the conversation as a team member	Intent Recognition
Produce Text	Add information to the conversation as a team member	GPTJ

The two main goals as shown in Table 1 are moderating speaking equity and adding relevant information to the conversation. The goal of moderating speaking equity will be done in parallel to the goal of adding relevant information to the conversation. Both goals serve the larger theme which is attempting to improve team decision making. Moderating speaking equity will be done by a python script which will be further explored in this section. Adding relevant information to the conversation will be done by a three-step process which includes detecting topic and ranking through the intent recognition methods similar to those explored in the literature review. The third step of this process will be implementing GPTJ to generate text.

3.1 Moderating Speaking Equity

This aspect of the bot will run parallel to the aspect that attempts to add information as a team member. The data features that are going to be utilized for this aspect of this goal are participant and duration. Duration, as described previously, is simply the duration the participant was speaking for that given sentence in the transcript. Differing time windows will be

implemented to moderate the discussion here. For example, in a 30 second time window it can be observed who is speaking the most and who is speaking the least. These will be incorporated as sliding time windows meaning they would be inserted at various points in the conversation. Four different time window lengths will be incorporated and tested to see how the results change when each of the four are used. The four different time windows that will be evaluated are 15 seconds, 30 seconds, 45 seconds, and 120 seconds. It will be evaluated if the person who may be over speaking in the 15 second window will also be over speaking in the 30 second and 45 second windows. Being able to observe these consistencies will elucidate if the time windows are too long or too short. If the shorter time windows show a significant difference in who is over or under speaking in each time window, the time windows are likely too short. It would be concluded that these windows are too short because if a different participant were over speaking in each window, then it is likely that the conversation is balanced over a longer period. Conversely if the over and under speaking distributions are equally lopsided for a short and a long period of time, the bot should intervene sooner rather than later to curb the chances of the conversation being dominated. The algorithm that will be used for the bot to decide who is speaking to much or too little is: if top speaker is speaking more than 70 percent of the time, top speaker will receive a prompt; if bottom speaker is speaking less than 20 percent of the time, bottom speaker will receive a prompt. This will all be programmed with a python script.

3.2 Intent Recognition for Topic Detection

Research about intent recognition will be applied to topic detection. The reason that it is necessary to make this slight distinction in terminology is that topic detection is more specific to this particular goal of the research. The other task involving intent recognition will be ranking detection. This will be implemented as a classification algorithm since each sentence is being

classified into its respective topic. The algorithm will be applied to leverage the sequential nature of the conversation so it can accurately detect which topic the group is discussing using contextual knowledge of previous sentences spoken instead of independently analyzing each sentence. For this reason, the model will utilize a recurrent neural network (RNN). The words in the text corpus will be tokenized using the Keras Tokenizer API since it is more computationally efficient than one-hot encoding. The RNN that will be used will consist of multiple layers which will includes an embedding layer, a Long Short-Term Memory (LSTM) layer, a dropout layer, and a dense layer. The decision has been made to include a LSTM layer in order to allow the algorithm to remember the context of the conversation more effectively, further leveraging the sequential nature of the data. The activation function that will be used is a SoftMax function with 16 output layers. For multi-classification problems, a SoftMax function is best suited since it outputs probabilities for all possible classifications; the classification with the highest probability will be chosen as the classification. The optimizer that will be used is an Adam optimizer and the metric that is being tested for is accuracy. The network will be trained over 200 epochs.

The layer hyperparameters will be experimented with for accuracy. The hyperparameters that will be tested are dropout levels of 0.25, 0.5 and 0.75. The LSTM hyperparameters will experiment with number of units, 25 and 50. These combinations of hyperparameters will create 6 unique RNN models per dataset. The model which achieves the highest accuracy will be used in the final model. These combinations will also be tested for in each of the three versions of dataset cleaning methods: dropping NA values, keeping NA values, and filling NA values with the most recently discussed item. The topics that will be detected are all 15 survival items plus an NA for times the group not talking about any specific item. These items include: Compress kit (with 28 ft. of 2-inch gauze), Ball of steel wool, Cigarette lighter without the fluid, Loaded .45-

caliber pistol, Newspaper (one per person), Compass, Two ski poles, Knife, Sectional air map made of plastic, 30 feet of rope, Family-sized chocolate bar (one per person), Flashlight with batteries, Quart of 85-proof whiskey, Extra shirt and pants for each survivor, and a Can of shortening.

3.3 Intent Recognition for Ranking Detection:

Ranking detection will be an algorithm similar to topic detection. The reason ranking detection is included is so that the model can have an accurate representation of the decisions being made and can respond with relevant information about the ranking number the group is discussing. The algorithm itself will also be similar to topic detection in that it is a classification algorithm using sequential data. This model will implement a recurrent neural network with a long short-term memory layer for this task and will train on the ranking feature that has been manually coded into the data file. This algorithm will classify the sentence into either a number 1 through 15 or it will be NA where the group members are not detectibly talking about a particular ranking. The layer hyperparameters and activation functions of the RNN will also be experimented for using the same methodology as in topic detection.

3.4 Using GPTJ

The detected topic and ranking will be collected and used as input for GPTJ. GPTJ stands for Generative Pre-trained Transformer, and it is a text processor that is built to resemble GPT-3 by OpenAI. The major difference between the two is that GPTJ was created by Eleuther AI and is fully open source, so there will not be a need to obtain any licensing from a third party in order to use the software. GPT-3 also reserves the right to terminate the project and all the data that is being worked on, so for that additional reason the decision has been made to opt for a software that does not have that ability. GPTJ is built using 6 billion parameters and is used to

create text that resembles human speech. This is what will be used to get the bot to speak into the conversation and act as a team member. GPTJ takes in a context, a prompt, and a number of examples, and then returns an output. The context is a short description of the type of response desired. For example, if someone were to program GPTJ to write a song then the programmer can write the context: “write a song”. The prompt is a set of text the algorithm should directly respond to. In the song example, a programmer may provide a part of a verse and then GPTJ will generate an output that would complete the verse. The example section provided to GPTJ is simply an example of a prompt and an example of the desired output. GPTJ also allows for setting character limits for the response. A character limit in the range of 40 and 200 will be set, however, to avoid words being split in the final response generated, the response will be truncated to the first full sentence generated. It is also possible to tune a parameter of GPTJ called temperature. Temperature is essentially the randomness level of what the response will be. The benefit of turning this parameter higher is that there can be more creative and interesting responses from the algorithm. The benefit of keeping the temperature lower, however, is that the response will be more predictable in that it will stay closer to the example and context. The potential downside of keeping the temperature low is that the response could get stuck in loops. In this scenario, GPTJ only finds a small subsect of words that is close enough to the prompt so it will keep repeating those words. It is recommended to keep the temperature up high enough to avoid this response. If the temperature is kept too high, however, the risk is that the response could get too random and unrelated to the prompt. The temperature will be set at 0.75 to attempt maintaining an optimal level ideally avoiding both randomness and repetition.

GPTJ will be used to produce text to attempt to add relevant information as if it were a team member in the conversation. For this to properly function, it is likely necessary to

accurately identify the topic and ranking being discussed. There will be custom prompts and contexts for each topic to maximize the relevancy of the information given. The bot should respond with two things: knowledge about the topic at hand and also what the bot would put as the correct ranking. The process is not hard coding the correct answers being output but giving GPTJ the context of what the right answer is so it can generate relevant speech. For this functionality, it is necessary to implement two different contexts and prompts for each part of the response. For inserting knowledge about the topic, the context would be assigned as the expert information given about the topic at hand. The Winter Survival Task manual comes with this expert knowledge in the key so there would be no need for a third-party source for this. The expert knowledge given for flashlight, for example, is: "6. Flashlight. Inasmuch as the group has little hope of survival, if it decides to walk out, its major hope is to catch the attention of search planes. During the day the lid mirror, smoke, and flags made from clothing represent the best devices. During the night the flashlight is the best signaling device. It is the only effective night-signaling device besides the fire. In the cold, however, a flashlight loses the power in its battery very quickly. It must, therefore, be kept warm if it is to work, which means that it must be kept close to someone's body. The value of the flashlight lies in the fact that, if the fire burns low or inadvertently goes out, the flashlight could be immediately turned on the moment a plane is heard." This would be provided as the context for flashlight. The topic selected would be a variable and there would be contexts similar to this defined for each item in the survival list. With this context, GPTJ would be given a prompt of "Summarize this article," with the context given. With that context and prompt, GPTJ can ideally generate a meaningful response about the topic that the group is talking about.

For the second part of the response, GPTJ will execute a similar process. Instead of using the output from the algorithm to detect topic for selecting context, the algorithm for detecting ranking will be used. If the ranking algorithm detects 8, for example, then the expert knowledge for the correct ranking 8 will be used as context. Using what is given in the Survival Key it would be as follows: “8. Newspaper (one per person). The newspaper could be used for starting a fire much the same as the rope. It will also serve as an insulator; when rolled up and placed under the clothes around a person's legs or arms, it provides dead-air space for extra protection from the cold. The paper can be used for recreation by reading it, memorizing it, folding it, or tearing it. It could be rolled into a cone and yelled through as a signal device. It could also be spread around an area to help signal a rescue party.” Contexts such as these will be given for each item in the survival guide. The prompt would be similar to the topic prompt as well, however, the output would be slightly different. The response for this section will be prefaced with: “What I think about ranking number x is,” and then the output would be concatenated with that sentence as a String. The reason there is a need to split the code into two different sections is so that GPTJ has the correct contexts for each survival item it needs to generate text for. If the contexts were put together, then the response would likely be nonsensical and generating text about both items without a clear defining line on what it is talking about. Splitting the GPTJ algorithm into two parts ensures that a relevant context is given, and that the response stays on topic.

3.5 Architecture

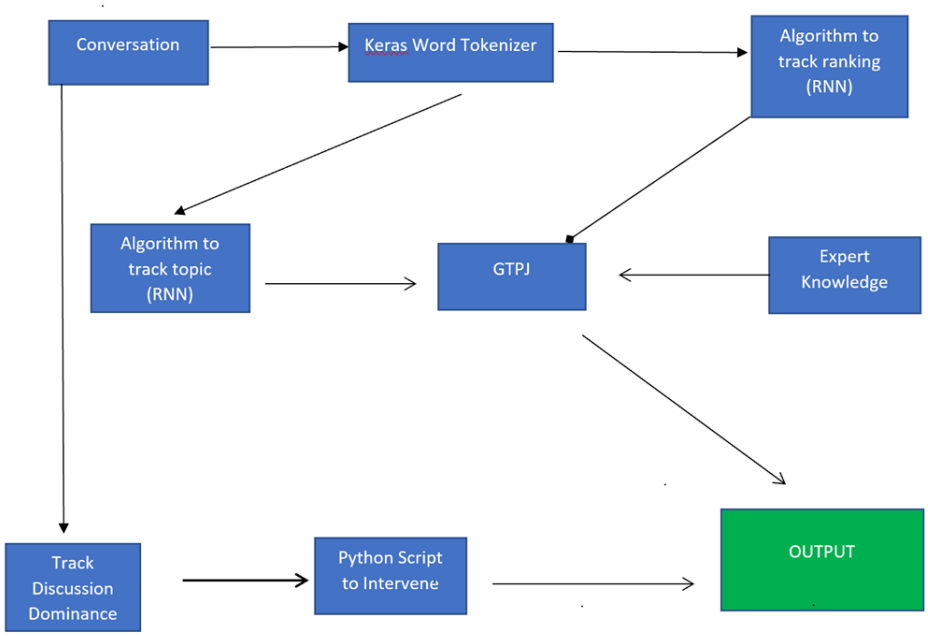


Figure 3: Architecture Map

Figure 1 provides a visual representation of the architecture proposed for the model. As seen, the conversation will be processed in two routes. It will be processed into tracking discussion dominance, and it will be processed into a TensorFlow Keras word tokenizer for the speech generative portion of the bot. When it is processed into tracking the discussion dominance, no word tokenization or processing will be needed. It will simply record speaking durations for each participant in each time frame. This will be executed by a python script previously discussed in section 3.2. Once the durations are recorded and calculated, this data will go into the decision script to intervene. This method will decide if any team member is over speaking or under speaking and then it will select the appropriate person to prompt if it needs to prompt at all.

The other direction the conversation data will be processed in is the generative text side for adding relevant information. This will begin with all the sentences in the data being tokenized by a TensorFlow Keras word tokenizer. Once the sentences are ready it will be used as input for two separate algorithms. One will be for topic detection and the other for ranking detection. Each of these algorithms will be a Recurrent Neural Network as described previously in this section. Once the RNNs output the resulting topic and ranking, these outputs will be used as inputs for the appropriate GPTJ algorithm which will be given the correct expert knowledge from the winter survival task guide. When GPTJ has both elements with its correct context and prompt, it can then develop an intelligible response which will then be inserted into the conversation as an output.

3.6 Data

The Group Affect and Performance (GAP) Corpus has been collected at University of the Fraser Valley (UFV, Canada). The GAP Corpus uses a winter survival task scenario, with participants first completing a ranking task individually and then performing the same ranking task jointly as a group. The full set of instructions for the task can be found in Appendix A. All of the group conversations are in English. The corpus consists of 28 meetings of discussions of the Winter Survival Task after they have each completed the task individually. These 28 meetings were conducted in person and then transcribed into text for the purposes of data processing. The meetings themselves make up over 4 hours of speech time which translates into roughly 70 thousand words.

The transcripts in this section were annotated with a number of labels. Each sentence was transcribed and had the possible labels of Participant, Start, End, Duration, Sentiment, Decision, Private and Survival Item variables. The ‘participant’ variable also has a number before and a

number after the color name. The number before signifies the group number, 1-28, and the number after is the number of times they have spoken so far. So, for example, if participant Green in group 3 is speaking for the 8th time then the participant label for that sentence would be 3.Green.8. The ‘Start’ variable denotes the time, to the millisecond, that the participant begins his/her utterance. The ‘End’ variable denotes the time, to the millisecond, that the participant ends his/her utterance. The ‘Sentence’ variable is what the participant said. Sentiment can be positive or negative, and it signifies the sentiment of the sentence uttered by the participant. Decision can be Proposal, Agreement, Disagreement, or Confirmation and represents if any of those decisions were signified by the sentence spoken at that time. Private represents the decision that each individual made. In other words, this means what the individual decided on their individual task before they decided on it as a group. Survival Item denotes which survival item they are discussing.

It will also be necessary to also manually add in another column that will help with the agent and that is a ranking column. The ranking column will represent which ranking the team is currently talking about. The coders will go through that data and label each sentence with the appropriate ranking that the team members are currently discussing, much like the survival item is labeled for which item they are currently discussing. Adding this column will give the agent more context to talk about as discussed in the ranking detection section.

The annotation, while providing a large number of variables, also has a significant number of missing values as shown in table 2.

Table 2: Missing Values

Column Name	Number of Values Missing	Percent of Values Missing
Sentiment	7149/8009	89.2%

Decision	6508/8009	81.2%
Private	7277/8009	90.8%
Survival Item	5119/8009	63.9%

Since the data contains a significant number of missing variables, it is necessary to validate this missing information. The fact that the variables are missing is not an inherent issue, however, it does become an issue if it is mislabeled. A correctly labeled missing value does not signify that the data are missing information, rather it provides its own information. If the data are missing a value on private, it simply means that the sentence was not from someone's private list. If the data are missing a value from decision, it simply means that the sentence in question was not making any proposal, confirmation, agreement, or disagreement to any other team member. This logic applies to all other features as well. It is useful and necessary, however, to do this validation and assess the accuracy of the data that is being handled.

A team of two coders will manually parse through the data and ensure that each label is correct. Each individual coder will add the ranking labels where appropriate and verify the accuracies of survival item features. These are the two features that are essential for the topic and ranking detection tasks. The participant and duration features are essential for the agent moderating speaking equity, but those have been transcribed from an audio set by the researchers that made the dataset so there is not much there can be done to review. There are also no missing values for these columns, so it is likely that the correct labeling of the transcript has been done by the researchers that developed it.

The data verification process will entail each coder to first verify the data individually. These verified files will then be compared using a python script, and all detected differences will

then be reviewed and decided upon by consensus. For example, if one coder were to put Rope in as the survival item and the other coder were to leave the row for the column blank, the python script would detect this difference and the coders will then reassess the sentence and come to a definitive consensus about the final labeling. The verified values have the distribution as shown in table 3.

Table 3: Value Distribution

Ranking	Count	Survival Item	Count
1	971	Air Map	208
2	863	Shortening	205
3	572	Compress Kit	178
4	567	Compass	177
5	590	Whiskey	174
6	516	Cigarette Lighter	171
7	601	Pistol	169
8	475	Flashlight	163
9	484	Chocolate Bar	158
10	459	Newspaper	156
11	487	Knife	156
12	259	Steel Wool	153
13	291	Ski Poles	145
14	382	Rope	144
15	479	Shirt/Pants	133

As shown in table 3, there is still a significant number of missing values in ‘Survival Item’ after verification. Different variations of handling missing data will be experimented with to gain the highest possible accuracy. Survival Item with the verified n/a values included, with the n/a values removed, and with the n/a values filled with the most recent item discussed. The large amounts of missing data could also be handled by balancing the dataset. This method, however, would undermine the sequential aspect of the research that is being examined. The LSTM layers in the RNN are leveraging the sentences being said in order and not individually as sentences; balancing the dataset may disrupt this order. It is instead better to test a dataset

cleaning method where the NA values are simply dropped so at least the rest of the data stays in order. The dataset with the highest performing RNN configuration will be used in the final model. This will not apply for the duration feature or ranking feature since there are no missing values.

CHAPTER 4

EXPERIMENTATION AND TESTING

A separate experimentation method is necessary for each aspect of the research. A testing method will be performed for moderating speaking equity, topic detection, ranking detection, and for speech generation. Each meeting file will be run individually for each portion of the experimentation.

4.1 Moderating Speaking Equity

For moderating speaking equity, the testing method needs to assess whether the right people are being prompted at the right times. To ensure the decision algorithm is working, the experimentation should spot check 10 instances across all conversations for each time window and ensure the algorithm is prompting participants correctly. This process entails manually calculating the duration for speech for each participant in the time window and verifying that the code is also calculating the same number. The process also involves verifying the algorithm is then prompting the right people based on the decision algorithm put in place. This portion of experimentation is simply ensuring the decision algorithm is implemented properly. The number of samples that will be taken here is limited because it can be safely assumed that if the code is working in a few instances then it will work across all conversations for this simple functionality. The methodology should also elucidate how conversation dominance appears in each differing time window. For comparing interventions across differing time windows, the testing method can examine each time window across each conversation and compile a list of who gets prompted for over and under speaking and these lists can then be compared. If it is observed that

the same people are getting prompted in each window, then it may be possible that the distribution of speaking is enduring throughout the entirety of the conversation. Furthermore, it can be concluded that a more frequent intervention method may be necessary so the distribution of speaking can be balanced as soon as possible. Conversely, if the discussion dominance is different across time windows the conclusion can opt for a longer time window because it can be observed that the distribution of speaking is not as enduring. This means that whoever may be dominating in a 15 second time window may not be dominating in the 45 second time window and others are being given the chance to speak. In this case, the conversation does not require an intervention in the 15 second time frame because the conversation is naturally being equalized.

4.2 Topic and Ranking Detection

For topic and ranking detection, the experimentation will include the procedure described in the proposed model. The best-performing model of the 6 variations of neural networks will be selected by testing for the accuracy of each model fitted on 5 of the same randomly selected meeting files. The experimentation procedure will then fit the best performing model on 25 randomly selected meeting files and will generate an accuracy score. Three files will be left out for evaluation which will include GPTJ in the complete model. The metric that will be used for this is an accuracy percentage. The blank values will be labeled as N/A and the experimentation will see if the algorithm would work despite the missing data. A minimal amount of manipulation of the data will also be done to experiment with the accuracy of these neural networks. A version of topic detection will be used with the original verified dataset which will include n/a values for the topic. Another version will have all n/a values filled will be tested with the same network. This will only be done for the topic detection network as the ranking detection will have not have as many n/a values. The missing values will be filled in with the most recent

survival item discussed until there is another topic labeled in the verified data. With that, the research would be able to generate an accuracy for how well these algorithms are working. All variations of the neural network mentioned in proposed model will be tested and the network with the highest accuracy will be used as the network in the final model. This resulting network will be used to identify the topic and ranking and propagate this information into GPTJ for creating intelligible text.

4.3 Intelligibility of Responses

The methodology for assessing the intelligibility of the responses will require a separate process. A certain number of points where the bot would intervene in the conversation will be established. These points of intervention should come when the group is at varying points of conversation. The test will observe how well the bot responds when the group is at a clear topic in conversation, when the group is talking about multiple topics in conversation, and when the group is not talking about any clear topic of conversation involving a survival item. The method will aim to examine each of these points of conversation to observe the versatility of the bot. It will be of most interest to observe how it responds to a clear topic being discussed since this is the normal case where a group would most likely prompt the bot. For this reason, the testing method will have 60 percent of the intervention points take place at these points and 20 percent each for the other two type of intervention points. The methodology for assessing the intelligibility of the bot's response will include a team of four research assistants to code through the responses.

Once the points of intervention are selected, the method will have a response from the bot at each point. Once these responses are obtained, the research assistants will indicate how relevant and intelligible the response is based on three questions on a 5-point scale, 1 being the

lowest and 5 being the highest. The three questions are: is this relevant to the topic/ranking being discussed, does the response sound like something a person would say, and does this response help the team's decision making. Once each individual team member gives their ratings, the ratings will be averaged to give each response its final rating. These ratings will also include the degree of agreement among the responses. The agreement will be represented by the standard deviation of the set of ratings given for each response. The bot will be given an overall score with the average of all the ratings for all the responses including the average rate of agreeability. The testing method will elucidate the ratings for all interventions with a clear topic, interventions between topics, and interventions with no clear topic combined. These ratings will be considered when judging the overall performance of the bot.

CHAPTER 5

RESULTS AND DISCUSSION

5.1 Moderating Speaking Equity

Table 4: Speaking Distribution

Mean Speaking Time	Standard Deviation of Mean	Largest Difference in Min and Max Speaker	Smallest Difference in Min and Max Speaker	Mean Difference between Max and Min
183.53s	119.29s	478.5s	13.1s	155.6s

Table 5: Prompt Distribution

Time Window	15	30	45	120
Average % Prompted (over)	31.4%	16.7%	12.3%	8.5%
Average % Prompted (under)	74.3%	65.7%	63.3%	59.3%
SD of Average (over)	17.6%	18.1%	20.6%	25.7%
SD of Average (under)	27.6%	32.2%	35.1%	39.1%
Confidence Interval(over)	(25.7%, 37.1%)	(10.8%, 22.5%)	(5.6%, 18.9%)	(0.3%, 16.8%)
Confidence Interval(under)	(65.4%, 83.2%)	(55.3%, 76.1%)	(52.0%, 74.6%)	(46.6%, 71.9%)

The mean speaking time per person per conversation is 183.5 seconds with a standard deviation which is at 119.29 seconds. This suggests the speaking distributions of these conversations have a significant amount of variation. Some conversations have a nearly perfect, even distribution among participants, such as the conversation seen with a 13 second difference

between the most active and least active speaker. Other conversations, conversely, have an entirely lopsided distribution as seen in the conversation where the most active speaker spoke for 478 seconds more than the least active speaker. While these large differences in speaking distribution exist, it is still useful to view how different time windows prompt speakers across all conversations.

The frequency of prompts, as seen in table 2, reduces as the time windows expand. This suggests that over time, speaking distributions in groups tend to become more even. The 15 second window prompts members the most frequently, which could be attributed to the fact that all members may not have sufficient time to speak within that small of a window. The average speaking duration across all conversations, however, is 1.93 seconds for each person. Even with this small average speaking time, a 15 second window may be triggering a prompt too often. The difference in percentage of prompts is large between 15 and 30 seconds but becomes a much more stable difference between 30 and 45 seconds despite the three time windows being evenly spaced. This suggests that the distributions of time that are too lopsided in a 15 second time window become more evenly distributed once a 30 second window is examined and remains relatively stable even throughout a 120 second time window. The possible ail of using a large window such as the 120 second window is the possible loss of information. A large window may lose information about who is over of under speaking within a smaller amount of time and would not be able to detect this discrepancy in participation. The large window, when implemented, may also take too long to prompt a person who is over or under speaking as compared to a shorter window that detects the same participation discrepancy earlier.

5.2 Topic and Ranking Detection

Table 6: Ranking Accuracies

Model	Training Accuracy	Validation Accuracy (top answer)	Top-3
Dropout: 25%, LSTM layers: 25	76.92%	13.15%	39.41%
Dropout: 50%, LSTM layers: 25	**72.49%**	**16.13%**	**43.50%**
Dropout: 75%, LSTM layers: 25	56.51%	12.90%	37.64%
Dropout: 25%, LSTM layers: 50	78.40%	13.90%	38.23%
Dropout: 50%, LSTM layers: 50	79.29%	13.90%	40.00%
Dropout: 75%, LSTM layers: 50	69.23%	14.39%	40.58%

Table 7: Survival Item Drop Na Values

Model	Training Accuracy	Validation Accuracy (top answer)	Top-3 Accuracy
Dropout: 25%, LSTM layers:25	**100%**	**49.22%**	**61.20%**

Dropout: 50%, LSTM layers: 25	92.59%	48.44%	59.68%
Dropout: 75%, LSTM layers: 25	65.74%	43.75%	65.89%
Dropout: 25%, LSTM layers: 50	100%	46.88%	60.46%
Dropout: 50%, LSTM layers: 50	96.30%	49.22%	58.13%
Dropout: 75%, LSTM layers: 50	87.96%	48.44%	62.79%

Table 8: Survival Item with Na values

Model	Training Accuracy	Validation Accuracy (top answer)	Top-3
Dropout: 25%, LSTM layers:25	**98.77%**	**73.08%**	**79.61%**
Dropout: 50%, LSTM layers: 25	96.63%	70.38%	77.69%
Dropout: 75%, LSTM layers: 25	83.44%	62.31%	76.53%
Dropout: 25%, LSTM layers: 50	99.39%	68.46%	76.53%

Dropout: 50%, LSTM layers: 50	97.85%	69.62%	78.84%
Dropout: 75%, LSTM layers: 50	92.33%	69.23%	75.76%

Table 9: Survival Item Filled Values

Model	Training Accuracy	Validation Accuracy	Top-3
Dropout: 25%, LSTM layers:25	**77.10%**	**36.08%**	**46.67%**
Dropout: 50%, LSTM layers: 25	73.23%	34.51%	44.31%
Dropout: 75%, LSTM layers: 25	60.65%	29.41%	43.52%
Dropout: 25%, LSTM layers: 50	78.39%	35.29%	45.88%
Dropout: 50%, LSTM layers: 50	76.45%	28.24%	44.31%
Dropout: 75%, LSTM layers: 50	68.71%	34.90%	48.62%

For topic detection, the hyperparameter combination with the highest validation accuracy is a dropout rate of 25% with 25 LSTM layers. This combination generally achieved among the highest training accuracies for each type of dataset cleaning method trained. It is also important to note that the training accuracies reported are from the last epoch trained, so some may be

reported as 100% simply because the last epoch was trained to 100%. The validation metric, however, is the deciding factor in which combination is selected for training the final model since it is the best indicator of how generalizable the algorithm is to other conversation transcripts. Out of the three data cleaning methods, the model that included all original NA values performed the best by a large margin. Choosing this dataset, however, does entail a few important caveats. It is possible that the reason the accuracy is so high is because it is simply guessing NA for a large section of the predictions. When considering only non-zero accuracies, the best model of this dataset only has a 30.2% accuracy. While many of the NA predictions may be correct, it does not assist in identifying when the group is talking about certain identified topics. This also poses additional issues to the model because feeding an NA value into the response generation algorithm will not produce any meaningful text that will assist the decision-making process. For these reasons, choosing the second-best performing dataset of the model may prove more useful. The second-best performing dataset is dropping the NA values. The best performing model in this dataset produces a near 50% validation accuracy. While this accuracy is certainly lower than the 70% accuracy produced by the aforementioned model, it is guaranteed that none of the produced classifications will be NA, so the response generation will always be provided with a topic.

For ranking detection, a dropout level of 50% was preferred. The best performing model also included 25 hyperparameter layers. This will be the model that gets used in the final model. It should also be noted that the validation accuracies are generally low. This may be due to the nature of the labeling of the data. When the ‘Ranking’ column was originally labeled by the research assistants, there was a roughly 35% disagreement rate. Since this high discrepancy

existed when analyzing the data, it could be assumed that assigning the correct ranking label through a conversation transcript may be a fundamentally difficult task.

The model does, however, generate a higher accuracy when considering the top 3 closes correct rankings instead of just the top ranking. The top 3 ranking considers if the correct answer is in the top three guesses that the algorithm makes. It is found that often the top ranking selected by this algorithm may be incorrect, but the selected network generates a 43.5% accuracy when it considers the top three rankings that the correct answer may be. Only the top ranking will be passed to GPTJ, but for the purposes of evaluating the accuracy of the model, a top-3 ranking adds a bit of information. Similarly, for the topic section, when considering top-3 instead of the top 1, the accuracy increases from around 50% to 61.2%. There do exist higher top-3 accuracies in the other models, however, these should not be considered when selecting the best performing model for the top answer. The top 3 ranking is a metric to simply show the general accuracy of the RNN models that will be used in the final model. The top answer will be the only answer that will be passed on to GPTJ, so the validation accuracy for the top answer is the main metric considered for selecting a model.

The training data for all models also achieved relatively high accuracies compared to the top 1 validation data, however, this could be a result of overfitting a small dataset. Generally, to combat overfitting in a small model, it is best to reduce the size of the neural network. This assumption is supported in the findings since the models with 25 layers consistently outperformed the models with 50 layers. All of the best performing models also did outperform a benchmark algorithm, which was a multinomial naïve bayes model. The benchmark predicted ‘Survival Item’ with an accuracy of 10.8% and ‘Ranking’ with an accuracy of 9.4%, both of which were outperformed by the RNN models.

Table 10: Best Models Selected

Task	Best Model	Validation Accuracy	Top 3 Accuracy
Item Detection	Drop NA dataset, Dropout:25, LSTM: 25	49.22%	61.2%
Ranking Detection	Dropout:50, LSTM: 25	16.13%	43.5%

5.3 Generated Responses

Table 11: Generated Text

	Relevance to Topic	Relevance to Ranking	Does this sound like something a person would say	Does this help with team decision making
Average	1.92	1.18	2.94	2.09
Standard Deviation	1.68	0.84	1.99	1.77

The model faced a few challenges regarding performance. For generating a response that was relevant to the topic, it was limited to using topics that had a 50 percent accuracy. The model was then to generate a response based on that limited accuracy. This limitation was observed even more drastically for generating a response that was helpful for the ranking, which had a less than 20 percent accuracy from the RNN model. The temperature setting for GPTJ was set at 0.75, which resulted in some responses that were merely repetitions of the context fed into the model and some responses that were almost entirely random. Since GPTJ was given the task of summarizing the context and using that to generate a response, it may be assumed that the context was simply not long enough to generate a summarization that was very different from the original. In the instances that were more random, there still seems to be a semblance of connection to the context given, but that is not apparent by observing the response alone. For

example, response 2 generated text which stated ‘this article is not medical advice’ which does not make any sense in the context of the conversation. This does, however, make a bit of sense when it is considered that the topic recognized was the compress kit which has a medical context. In this instance, even though the topic was correctly recognized by the RNN model, the response generated was still irrelevant to the topic being discussed.

Improvements for this model could include using a more accurate topic/intent detection algorithm. The architecture of the GPTJ model could also be optimized. Instead of tasking the model to summarize a small set of domain relevant information, there could be a more intensive fine-tuning process deployed. This would require a much larger dataset that was used in this study and would require a more computationally intensive process but could possibly yield better results.

Although the model produced overall results that were arguably something a person would say, there exists a bit of disagreement among evaluators for this statement exemplified by the standard deviation among responses. It is also important to note that the sample size of these responses is relatively small. In order to gain a more robust exploration of this architecture’s capability, it is important to conduct a larger evaluation initiative.

CHAPTER 6

CONCLUSION

6.1 Future Work

Future work in this domain can be expanded in multiple ways. The implementation itself can be expanded into a real conversation where all mechanisms are deployed in a conversation in real time. This can be possible over any platform that has an automatic transcription feature where multiple people are speaking. The topic and ranking detection could analyze the conversation in real time instead of over the entire conversation transcript. The response generation aspect would take the results of the topic/ranking detection and also generate a response and interject live in a conversation. This methodology would be able to observe how the team's decision-making ability would react in a real scenario with the added information. The same can be done for moderating speaking equity. The four time-windows explored could be implemented into a team in real time and data could be collected on how quickly and effectively speaking distributions would shift. The conducted research explores an extrapolation of what the implementation of these prompting time windows may do, however, real time changes in speaking distributions are yet to be examined using this framework.

The performance of topic detection can also be improved upon by experimenting with a larger series of different architectures. These architectures may or may not include a Recurrent Neural Network model. It is possible to experiment with different word encodings as well in order to generate a possibly more accurate result. It was observed that smaller networks performed better with smaller datasets, so different combination of small networks could be

tested. Larger datasets could also be experimented with to analyze how the models would perform under different dataset sizes and dataset types.

Response generation in GPTJ could be tuned for greater accuracy. Different configurations including fine-tuning models could be implemented. Using a larger corpus of expert information may also produce different results and would be a worthwhile factor to consider.

6.2 Conclusion

The data on moderating speaking equity was tracked and analyzed for groups of 2 to 4 people executing one specific 15-minute task. These findings may be applicable to other team decision making domains and the speaking distributions may be indicative of a larger trend. The optimal time window for intervention could be tested for in a live group setting. Once implemented in a live group setting, each time window could be implemented, and a conclusion could be made about the effectiveness of the applied method. This could also possibly be expanded to groups that are larger than four people.

The tracking of conversation topic and ranking performed with a near 50% accuracy. This method of following a conversation may be generalizable to other conversation domains, however, there may be more accurate methods of achieving this process. With a more accurate intent recognition model for conversation group decision making, and a more tuned GPTJ architecture, the proposed model may still be able to assist in a group's decision-making process. This portion of the architecture could also be applied to a live group setting where response generation happens in real time.

In the domain of team decision making, it is concluded that there is a large potential for artificial intelligence to be inserted into this domain and possibly influence how decisions are

being made. The research conducted acts as a proof of concept for this potential in moderating information flow through tracking speaking frequencies and through inserting artificial intelligence into a team as a contributing member. There is promise for the future of integrating artificial intelligence into team decision making environments.

In the domain of artificial intelligence, the efficacy of using a recurrent neural network with LSTM layers for intent recognition in the domain of free-flowing conversation is determined. The overall architecture of generating speech in the process explored is also evaluated. The implemented model is one of many possible algorithms that could be inserted in this domain. Although performance was limited in terms of accuracy, a possibly relevant framework for achieving the goal of implementing AI into free-flowing conversation is provided.

References:

Biderman, S., & Raff, E. (2022). Neural Language Models are Effective Plagiarists. CoRR, abs/2201.07406. Opgehaal van https://arxiv.org/abs/2201.07406

Brodbeck, F. C., Kerschreiter, R., Mojzisch, A., & Schulz-Hardt, S. (2007). Group Decision Making under Conditions of Distributed Knowledge: The Information Asymmetries Model. The Academy of Management Review, 32(2), 459–479. https://doi.org/10.2307/20159311

Campbell, J., & Stasser, G. (2006). The Influence of Time and Task Demonstrability on Decision-Making in Computer-Mediated and Face-to-Face Groups. *Small Group Research*, *37*(3), 271–294. https://doi.org/10.1177/1046496406288976

Chahine, S., Cristancho, S., Padgett, J., & Lingard, L. (2017). How do small groups make decisions? *Perspectives on Medical Education*, *6*(3), 192–198. https://doi.org/10.1007/s40037-017-0357-x

Goksel Canbek, N., & Mutlu, M. E. (2016). On the track of Artificial Intelligence: Learning with Intelligent Personal Assistants. Journal of Human Sciences, 13(1), 592–601. Retrieved from https://www.j-humansciences.com/ojs/index.php/IJHS/article/view/3549

Goo, C.-W., Gao, G., Hsu, Y.-K., Huo, C.-L., Chen, T.-C., Hsu, K.-W., & Chen, Y.-N. (2018, Junie). Slot-Gated Modeling for Joint Slot Filling and Intent Prediction. Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers), 753–757. doi:10.18653/v1/N18-2118

Hogan, K., Baer, A., & Purtilo, J. (2021). Diplomat: A conversational agent framework for goal-oriented group discussion. *ArXiv:2105.05322 [Cs]*. http://arxiv.org/abs/2105.05322

Hong, S. (ray), Suh, M. (mia), Henry Riche, N., Lee, J., Kim, J., & Zachry, M. (2018). Collaborative Dynamic Queries: Supporting Distributed Small Group Decision-Making. In Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (bll 1–12). doi:10.1145/3173574.3173640

Indulkar, A. P., Varadharajan, S., & Nayak, K. (2018). *Neural Intent Recognition for Question-Answering System*. *5*(3), 12.

Joshi, M. P., Davis, E. B., Kathuria, R., & Weidner, C. K. (2005). Experiential Learning Process: Exploring Teaching and Learning of Strategic Management Framework through the Winter Survival Exercise. Journal of Management Education, 29(5), 672–695. https://doi.org/10.1177/1052562904271198

Kim, S., Eun, J., Oh, C., Suh, B., & Lee, J. (2020). Bot in the Bunch: Facilitating Group Chat Discussion by Improving Efficiency and Participation with a Chatbot. In Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems (bll 1–13). doi:10.1145/3313831.3376785

Kiesler, S., Siegel, J., & McGuire, T. W. (1984). Social psychological aspects of computer-mediated communication. American Psychologist, 39(10), 1123–1134. https://doi.org/10.1037/0003-066X.39.10.1123

Lee, S. (2018). Toward Continual Learning for Conversational Agents. *ArXiv:1712.09943 [Cs]*. http://arxiv.org/abs/1712.09943

Lorenc, P., Marek, P., Pichl, J. et al. (2021). Benchmark of public intent recognition services. Lang Resources & Evaluation https://doi.org/10.1007/s10579-021-09563-3

Matiana, S., Smith, J. R., Teehan, R., Castricato, L., Biderman, S., Gao, L., & Frazier, S. (2021). Cut the CARP: Fishing for zero-shot story evaluation. CoRR, abs/2110.03111. Opgehaal van https://arxiv.org/abs/2110.03111

McCann, D. W. (1992). A Neural Network Short-Term Forecast of Significant Thunderstorms, Weather and Forecasting, 7(3), 525-534. Retrieved Mar 12, 2022, from https://journals.ametsoc.org/view/journals/wefo/7/3/1520-0434_1992_007_0525_annstf_2_0_co_2.xml

Ravuri, S., & Stolcke, A. (2015). Recurrent neural network and LSTM models for lexical utterance classification. In Sixteenth Annual Conference of the International Speech Communication Association.

Vásquez-Correa, J. C., Guerrero-Sierra, J. C., Pemberty-Tamayo, J. L., Jaramillo, J. E., & Tejada-Castro, A. F. (2021). One System to Rule them All: a Universal Intent Recognition System for Customer Service Chatbots. CoRR, abs/2112.08261. Opgehaal van https://arxiv.org/abs/2112.08261

Yin, W., Kann, K., Yu, M., & Schütze, H. (2017). Comparative Study of CNN and RNN for Natural Language Processing. CoRR, abs/1702.01923. Opgehaal van http://arxiv.org/abs/1702.01923

Appendix A: Winter Survival Task Instructions

"You have just crash-landed in the woods of northern Minnesota and southern Manitoba. It is 11:32 A.M. in mid-January. The small plane in which you were traveling has been completely destroyed except for the frame. The pilot and co-pilot have been killed, but no one else is seriously injured. The crash came suddenly before the pilot had time to radio for help or inform anyone of your position. Since your pilot was trying to avoid a storm, you know the plane was considerably off course. The pilot announced shortly before the crash that you were eighty miles northwest of a small town that is the nearest known habitation. You are in a wilderness area made up of thick woods broken by many lakes and rivers. The last weather report indicated that the temperature would reach minus twenty-five degrees in the daytime and minus forty at night. You are dressed in winter clothing appropriate for city wear—suits, pantsuits, street shoes, and overcoats. While escaping from the plane, your group salvaged the fifteen items listed below. Your task is to rank these items according to their importance to your survival. You may assume that the amount of each item is the same as the number in your group and that the group has agreed to stick together."

The items that you have been tasked to rank are the following: Compress kit (with 28 ft. of 2-inch gauze), Ball of steel wool, Cigarette lighter without the fluid, Loaded .45-caliber pistol, Newspaper (one per person), Compass, Two ski poles, Knife, Sectional air map made of plastic, 30 feet of rope, Family-sized chocolate bar (one per person), Flashlight with batteries, Quart of 85-proof whiskey, Extra shirt and pants for each survivor, and a Can of shortening.

The correct ranking goes in order from first to last as follows: Cigarette Lighter, Ball of Steel Wool, Extra Shirt and Pants for each survivor, Family sized Hershey's bar, Can of Shortening, Flashlight, Piece of Rope, Newspaper, 45 Caliber Pistol, Knife, Compress Kit, Ski Poles, Quart of 85 Proof whiskey, Sectional air map made of plastic, Compass.

Appendix B: Code Citation

The GPTJ code will be adapted from: https://www.pragnakalp.com/gpt-j-6b-parameters-model-huggingface/

This code will be adapted for the purposes of this research, but structurally will retain many similarities to the code found in the link. The code will also be adapted for the testing and implementation of the various hyperparameters mentioned in the proposed model section as well as in other ways to be more suited for this research. This is not the full code, but this is a snippet of the modified code that is essential for response generation.

```
for p in pred_points:
    guess = pred26[p]
    guess_name = rank_decode(guess)
    context = get_context(guess_name)
    input_text = "Summarize the following article:\n" + context + "\nSummarized
    inputs = tokenizer(input_text, return_tensors="pt")
    input_ids = inputs["input_ids"]

    output = model.generate(
    input_ids,
    attention_mask=inputs["attention_mask"],
    do_sample=True,
    max_length=len(context) + 20,
    temperature=0.75,
    use_cache=True,
    top_p=0.9
    )

    output = tokenizer.decode(output[0])
    split = output.split('Summarized Article:')
    response = split[1]
    response_sent = response.split('.')
    first_response = response_sent[0]
    print('Full', output)
    print("\n")
    print('For rank, ', guess_name, ' I think that ' , first_response)
```

Appendix C: Code

Modifications of presented code have been made throughout various instances of training/testing and analysis

Moderating Speaking Equity

```
for f in files:

    dic["file{0}".format(i+1)] = pd.read_csv(f)
    i = i+1



def to_sec(var, data):
    #data = dic["file{0}".format(file)]
    for x in range(len(data)):
        text_split = data[var][x].split(":")

        minutes = text_split[0]
        seconds = text_split[1]
        minutes = float(minutes)
        seconds = float(seconds)

        minutes = minutes *60

        seconds = seconds + minutes

        data[var][x] = seconds



def to_part(data):
    for p in range(len(data['Participant'])):
        if("Pink" in data['Participant'][p]):
            data['Participant'][p] = "Pink"
        if("Blue" in data['Participant'][p]):
            data['Participant'][p] = "Blue"
        if("Green" in data['Participant'][p]):
            data['Participant'][p] = "Green"
        if("Orange" in data['Participant'][p]):
            data['Participant'][p] = "Orange"
        if("Red" in data['Participant'][p]):
            data['Participant'][p] = "Red"
        if("Yellow" in data['Participant'][p]):
            data['Participant'][p] = "Yellow"
        if("Purple" in data['Participant'][p]):
            data['Participant'][p] = "Purple"
```

```
'''
times = []
def get_stats():
    file_times = []
    for s in range(1,29):
        file = dic['file{0}'.format(s)]
        membs = amount_membs(file)
        last = file['End'][len(file)-1]
        file_times.append(total_time_spoken(file, 0, last, membs))

    maxminDiff = []
    for time in file_times:
        #print (time, '\n')
        maxSpeak = max(time)
        minSpeak = min(time)
        diff = maxSpeak - minSpeak
        maxminDiff.append(diff)
    return stat.mean(maxminDiff)
```

```
def total_time_spoken(data, start_time, end_time, amount_membs):
    '''
    to_sec('Duration', data)
    to_sec('Start', data)
    to_sec('End', data)
    to_part(data)

    part_label = data['Participant'].factorize()
    data['Participant'] = part_label[0]
    #print ('amount of people ' , len(part_label[1]))
    '''

    part_0 = 0
    part_1 = 0
    part_2 = 0
    part_3 = 0
    part_4 = 0

    gen = (x for x in range(len(data)) if ((end_time>data['End'][x]) and (start_time<=data['Start'][x])))

    for x in gen:
            if(data['Participant'][x] == 0):
                part_0 = part_0 + (data['End'][x] - data['Start'][x])
            if(data['Participant'][x] == 1):
                part_1 = part_1 + (data['End'][x] - data['Start'][x])
            if(data['Participant'][x] == 2):
                part_2 = part_2 + (data['End'][x] - data['Start'][x])
            if(data['Participant'][x] == 3):
                part_3 = part_3 + (data['End'][x] - data['Start'][x])
            if(data['Participant'][x] == 4):
                part_4 = part_4 + (data['End'][x] - data['Start'][x])

    all_times = [part_0, part_1, part_2, part_3, part_4]
    times = all_times[:amount_membs]
    return (times)

def time_chart(start_time, end_time):
    blue, pink, green = total_time_spoken(start_time, end_time)
    timespoken = pd.DataFrame({'Time' :[blue, pink, green],
            'Participant': ['Blue','Pink', 'Green']})
    timespoken = timespoken.set_index('Participant')
    timespoken.plot.pie(y='Time', title = 'Time Spoken')
```

```
def intervene(data, start_time, end_time, amount_membs):
    under = None
    over = None
    times = total_time_spoken(data, start_time, end_time, amount_membs)
    tot_time_spoken = (sum(times))
    min_Speaker = min(times)
    max_Speaker = max(times)
    time = end_time - start_time
    min_index = times.index(min_Speaker)
    max_index = times.index(max_Speaker)
    #participant = ['blue','pink','green']
    if(min_Speaker <= tot_time_spoken*0.2):
        under = min_index
    if(max_Speaker >= tot_time_spoken*0.7):
        over = max_index
    return over, under

def amount_membs(data):
    to_sec('Duration', data)
    to_sec('Start', data)
    to_sec('End', data)
    to_part(data)

    part_label = data['Participant'].factorize()
    data['Participant'] = part_label[0]
    return len(part_label[1])


def windows(data, window):
    under_list = []
    over_list = []

    #data = dic['file3']
    membs = amount_membs(data)
    convo_len = data['End'][len(data)-1]

    point=0


    while (point+window<convo_len):
        over, under = intervene(data, point, point+window, membs)
        over_list.append(over)
        under_list.append(under)
        point = point+1
    over_nones = over_list.count(None)
    under_nones = under_list.count(None)
    over_prompts = (len(over_list) - over_nones)
    under_prompts = (len(under_list) - under_nones)
    over_perc = over_prompts/len(over_list)
    under_perc = under_prompts/len(under_list)
    return over_perc, under_perc
```

RNN Models

```
for f in files:

    dic["file{0}".format(i+1)] = pd.read_csv(f)
    i = i+1
```

```
def tokenize(file, drop):
    for q in range(1,29):
        data = dic['file{0}'.format(q)]
        #print('file{0}'.format(q))
        #print('Before:', dic['file{0}'.format(q)].shape)
        #pt_data = data[['Sentence', 'Survival Item']]

        if (drop == 'yes'):
            data = data.dropna(subset = ['Survival Item'])
        data = data.replace('Shirts/Pants','Shirt/Pants')
        data = data.replace('PIstol', 'Pistol')
        data = data.replace('Unknown', 'UNK/NA')
        data = data.fillna('UNK/NA')
        data = data.replace('Compress kit', 'Compress Kit')

        data = data[~data['Survival Item'].str.contains(",")]
        #print(data['Survival Item'])
        dic['file{0}'.format(q)] = data
        #print('after:', dic['file{0}'.format(q)].shape)
        #print((data['Survival Item'])[7])

    megafile = pd.DataFrame()
    for m in range(1,29):
            # read the csv file
        megafile = pd.concat([megafile, dic['file{0}'.format(m)]])

    pt_data = megafile[['Sentence', 'Survival Item']]

    pt_ints = pt_data.Sentence.values
    tokenizer = Tokenizer()
    tokenizer.fit_on_texts(pt_ints)
    vocab_size = len(tokenizer.word_index) + 1
    encoded_docs = tokenizer.texts_to_sequences(pt_ints)
    padded_sequence = pad_sequences(encoded_docs, maxlen=15)

    item_label = pt_data['Survival Item'].factorize()
    #print(item_label[0])
    sum =0
    for k in range(1,file):
        sum = sum + len(dic['file{0}'.format(k)])
    if (file ==1 ):
        ind_padded_sequence = padded_sequence[0:(len(dic['file{0}'.format(file)]))]
        ind_item_label = ((item_label[0])[0:(len(dic['file{0}'.format(file)]))])
    if (file ==28):
        ind_padded_sequence = padded_sequence[sum:]
        ind_item_label = ((item_label[0])[sum:])
    else:
        ind_padded_sequence = padded_sequence[sum:sum+(len(dic['file{0}'.format(file)]))]
        ind_item_label = ((item_label[0])[sum:sum+(len(dic['file{0}'.format(file)]))])

    #print(ind_item_label)
    return vocab_size, ind_padded_sequence, ind_item_label
```

```
def tokenize_rank(file):
    for q in range(1,29):
        data = dic['file{0}'.format(q)]
        data = data.dropna(subset = ['Verification'])
        dic['file{0}'.format(q)] = data

    megafile = pd.DataFrame()
    for m in range(1,29):
            # read the csv file
        megafile = pd.concat([megafile, dic['file{0}'.format(m)]])

    pt_data = megafile[['Sentence', 'Verification']]

    pt_ints = pt_data.Sentence.values
    tokenizer = Tokenizer()
    tokenizer.fit_on_texts(pt_ints)
    vocab_size = len(tokenizer.word_index) + 1
    encoded_docs = tokenizer.texts_to_sequences(pt_ints)
    padded_sequence = pad_sequences(encoded_docs, maxlen=15)

    item_label = pt_data['Verification'].factorize()
    #print(item_label[0])
    sum =0
    for k in range(1,file):
        sum = sum + len(dic['file{0}'.format(k)])
    if (file ==1 ):
        ind_padded_sequence = padded_sequence[0:(len(dic['file{0}'.format(file)]))]
        ind_item_label = ((item_label[0])[0:(len(dic['file{0}'.format(file)]))])
    if (file ==28):
        ind_padded_sequence = padded_sequence[sum:]
        ind_item_label = ((item_label[0])[sum:])
    else:
        ind_padded_sequence = padded_sequence[sum:sum+(len(dic['file{0}'.format(file)]))]
        ind_item_label = ((item_label[0])[sum:sum+(len(dic['file{0}'.format(file)]))])

    #print(ind_item_label)
    return vocab_size, ind_padded_sequence, ind_item_label
```

```
def make_rnn(intent, dropout_lev, lstm_layer, drop):
    for file in range(1,26):
        if intent == 'item':
            vocab_size, padded_sequence, item_label = tokenize(file, drop)
            #vocab_size9, padded_sequence9, item_label9 = tokenize(9, drop)
            vocab_size26, padded_sequence26, item_label26 = tokenize(26, drop)
            vocab_size27, padded_sequence27, item_label27 = tokenize(27, drop)
            vocab_size28, padded_sequence28, item_label28 = tokenize(28, drop)

        if intent == 'rank':
            vocab_size, padded_sequence, item_label = tokenize_rank(file)
            vocab_size26, padded_sequence26, item_label26 = tokenize_rank(26)
            vocab_size27, padded_sequence27, item_label27 = tokenize_rank(27)
            vocab_size28, padded_sequence28, item_label28 = tokenize_rank(28)
        embedding_vector_length = 32
        model = Sequential()
        model.add(Embedding(vocab_size, embedding_vector_length,
                                            input_length=15) )
        model.add(LSTM(lstm_layer, dropout=0.5, recurrent_dropout=0.5))
        model.add(Dropout(dropout_lev))
        model.add(Dense(16, activation='softmax'))
        model.compile(loss='sparse_categorical_crossentropy',optimizer='adam',
                                    metrics=['accuracy'])
        print(model.summary())
        history = model.fit(padded_sequence,item_label,
                    epochs=200, validation_split=0.1)
```

GPTJ

```
# loop over the list of csv files that shardul verified
for f in files:

    dic["file{0}".format(l+1)] = pd.read_csv(f)
    l = l+1


#Converts index in full file to index in predicted file
def find_index(ind, file):
    data = file
    data = data.dropna(subset = ['Survival Item'])
    data = data.fillna('UNK/NA')
    data = data[~data['Survival Item'].str.contains(",")]
    indicies = (data.index.tolist())
    #print (indicies)
    return((indicies.index(ind-2)))
```

```
#need to convert encoded items back to words
def item_decode(num):
    items = ['cigarette lighter', 'knife', 'compass', 'air map', 'whiskey',
          'steel wool', 'flashlight', 'compress kit', 'pistol', 'rope',
          'shirt/pants', 'UNK/NA', 'shortening', 'ski poles', 'chocolate bar',
          'newspaper']
    return items[num]

def rank_decode(num):
    ranks = [1.0, 2.0, 3.0, 4.0, 5.0, 7.0, 6.0, 8.0, 9.0, 10.0, 15.0, 14.0,
             13.0, 12.0, 11.0]
    return ranks[num]

def get_context(item):

    if item == 'cigarette lighter' or item ==1.0:
        return 'The gravest danger facing the group is exposure to the cold. The greatest need is for a source of warmt

    if item == 'steel wool':
        return 'To make a fire, a means of catching the sparks made by the cigarette\
                lighter is needed. Steel wool is the best substance with which to catch a spark and support\
                a flame, even if it is a little bit wet'
    if item == 'shirt/pants':
        return ' Clothes are probably the most versatile items one\
                can have in a situation like this. Besides adding warmth to the body, they can be used for\
                shelter, signaling, bedding, bandages, string when unraveled, and tinder to make fires.\
                Even maps can be drawn on them. The versatility of clothes and the need for fires,\
                signaling devices, and warmth make this number three in importance.'
    if item == 'chocolate':
        return 'To gather wood for the fire and to set up\
                signals, energy is needed. The Hershey bars would supply the energy to sustain the\
                survivors for quite some time. Because they contain basically carbohydrates, they would\
                supply energy without making digestive demands upon the body'
    if item == 'shortening':
        return 'This item has many uses the most important being that a mirror-like\
                signaling device can be made from the lid. After shining the lid with the steel wool, the\
                survivors can use it to produce an effective reflector of sunlight. A mirror is the most\
                powerful tool they have for communicating their presence. In sunlight, a simple mirror\
                can generate 5 to 7 million candlepower. The reflected sunbeam can be seen beyond the\
                horizon. Its effectiveness is somewhat limited by the trees, but one member of the group\
                could climb a tree and use the mirror to signal search planes. If the survivors have no\
                other means of signaling, they would still have better than 80 percent chance of being\
                rescued within the first twenty-four hours'
    if item == 'flashlight':
        return 'Inasmuch as the group has little hope of survival, if it decides to walk out, its\
                major hope is to catch the attention of search planes. During the day the lid mirror,\
                smoke, and flags made from clothing represent the best devices. During the night the\
                flashlight is the best signaling device. It is the only effective night-signaling device\
                besides the fire. In the cold, however, a flashlight loses the power in its battery very\
                quickly. It must, therefore, be kept warm if it is to work, which means that it must be kept\
                close to someones body. The value of the flashlight lies in the fact that, if the fire burns\
                low or inadvertently goes out, the flashlight could be immediately turned on the moment\
                a plane is heard.'
    if item == 'rope':
        return 'The rope is another versatile piece of equipment. It could be used to pull\
                dead limbs off trees for firewood. When cut in pieces, the rope will help in constructing\
                shelters. It can be burned. When frayed, it can be used as tinder to start fires. When\
                unraveled, it will make good insulation from the cold if it is stuffed inside clothing'
    if item == 'newspaper':
        return 'The newspaper could be used for starting a fire much the\
                same as the rope. It will also serve as an insulator; when rolled up and placed under the\
                clothes around a persons legs or arms, it provides dead-air space for extra protection\
                from the cold. The paper can be used for recreation by reading it, memorizing it, folding\
                it, or tearing it. It could be rolled into a cone and yelled through as a signal device. It\
                could also be spread around an area to help signal a rescue party'
    if item == 'pistol':
        return 'This pistol provides a sound-signaling device. (The international\
                distress signal is three shots fired in rapid succession.) There have been numerous cases\
                of survivors going undetected because, by the time the rescue party arrived in the area,\
                the survivors were too weak to make a loud enough noise to attract attention. The butt\
                of the pistol could be used as a hammer. The powder from the shells will assist in fire\
                building. By placing a small bit of cloth in a cartridge, emptied of its bullet, a fire can be\
                started by firing the gun at dry wood on the ground. At night the muzzle blast of the gun\
                is visible, which also makes it useful as a signaling device.'
    if item == 'knife':
        return 'A knife is a versatile tool, but it is not too important in the winter setting. It could\
                be used for cutting the rope into desired lengths, making shavings from pieces of wood\
                for tinder, and many other uses could be thought up.'
    if item == 'compress kit':
        return 'The best use of this item is to wrap the gauze around\
                exposed areas of the body for insulation. Feet and hands are probably the most vulnerable\
                to frostbite, and the gauze can be used to keep them warm. The gauze can be used as a\
                candlewick when dipped into melted shortening. It would also make effective tinder. The\
                small supply of the gauze is the reason this item is ranked so low.'
    if item == 'ski poles':
        return 'Although they are not very important, the poles are useful as a flagpole or staff\
                for signaling. They can be used to stabilize a person walking through snow to collect\
                wood, and to test the thickness of the ice on a lakeshore or stream. Probably their most\
                useful function would be as supports for a shelter or by the fire as a heat reflector.'
    if item == 'whiskey':
        return 'The only useful function of the whiskey is to aid in fire\
                building or as a fuel. A torch could be made from a piece of clothing soaked in the\
                whiskey and attached to an upright ski pole. The danger of the whiskey is that someone\
                might try to drink it when it is cold. Whiskey takes on the temperature it is exposed to,\
                and a drink of it at minus thirty degrees would freeze a persons esophagus and stomach\
                and do considerable damage to the mouth. Drinking it warm will cause dehydration. The\
                bottle, kept warm, would be useful for storing drinking water.'
    if item == 'air map':
        return 'This item is dangerous because it will encourage\
                individuals to attempt to walk to the nearest town—thereby condemning them to almost\
                certain death.'
    if item == 'compass':
```

```
#points26 = [10,27,44,51,125,149,65,58,77,101]
points26 = [10,27]
pred_points = []

for x in points26:
  pred_points.append(find_index(x, dic['file26']))


for p in pred_points:
    guess = pred26[p]
    guess_name = item_decode(guess)
    context = get_context(guess_name)
    input_text = "Summarize the following article:\n" + context + "\nSummarized Article:*"
    print(type(input_text))
    print(input_text)

    inputs = tokenizer(input_text, return_tensors="pt")
    input_ids = inputs["input_ids"]

    output = model.generate(
    input_ids,
    attention_mask=inputs["attention_mask"],
    do_sample=True,
    max_length=len(context)+50,
    temperature=0.8,
    use_cache=True,
    top_p=0.9
    )

    output = tokenizer.decode(output[0])
    split = output.split('*')
    response = split[1]
    print('Context: ', split[0])
    print('Full', output)
    print('Response' , response)
```