Application of Swarm Intelligence to Artificial Intelligences

Lois I. Omotara

Advisor: Bernard Chazelle

Independent Work Report, Fall 2023

Abstract

This project tested the efficacy of using swarming techniques to encourage group work amongst different chatbot programs powered by artificial intelligence. Swarming is an evolutionary strategy that enables simple organisms to correctly complete complex tasks and calculations and current technologies based on this behavior have seen success in optimizing computationally expensive tasks. Other than swarm.ai, which uses a physics-based model to optimize group work among humans, many of these techniques work by simplifying the individual components as much as possible. Therefore, I created a system based on the swarm.ai interface to simulate group work amongst different AIs to see if the answers generated by the group would be more "humanlike" and thoughtful than the answers of the individuals. Through testing I found that the model was only able to outperform one of the chatbots in 10 trials out of 40. This shows that group work amongst AI has the potential to develop more thoughtful AIs.

Introduction

In nature there exists a phenomenon known as swarm intelligence, and it is the evolutionary survival technique of organisms with limited intelligence. In terms of human behaviors, it can be thought of as an enhanced type of group work. Take bees, for example, when deciding on a new habitat there are many variables that must be considered in order to find the optimal home such as climate, availability of food, presence of competitions, personal preference, and other parameters. However, studies have shown that most of the types of the

colony are able to collectively decide on the most optimal location. This decision is made by the collective as each bee uses the iconic “waggle” dance to cast their votes. Another example includes the ant colony optimization problem which explains that ants are able to find the shortest distance to their food source due to the utilization of pheromones. This type of swarm intelligence is powered by a process known as stigmergy, in which a group of organisms are able to indirectly coordinate with each other using their environment. In both of these instances simple organisms are able to solve multivariate problems that are challenging even to humans and computers.

These impressive results beg the question: what could higher intelligences accomplish if they were able to utilize swarming as well. For humans, this is almost completely impossible, at least in the case of traditional swarming. The two previous examples show that traditional swarming is only possible due to the physiology of the two organisms that they gained through their unique evolutions. These examples, however, show the possible benefits of optimizing the human group-work to mimic swarm intelligence. It would allow for better business and political decisions that would maximize the benefit to the collective and minimize selfish incentives. However, due to the superior intelligence of humans and the ability to communicate verbally, humans did not develop this physiology.

The same is true for artificial intelligence that attempts to mimic neural processes. Although their mathematical and scientific knowledge far exceeds the ability of a single human being, there are still many drawbacks that keep them from being applicable to more sensitive matters. The main ones being their inability to consider social context, a general lack of common sense, and the lack of doubt in their responses. It is well known, however, that all pieces of artificial intelligence include an innate amount of bias due mainly to the biases of their creators

and the datasets they are trained on. Let's consider the possibility of applying swarm intelligence to different artificial intelligences. Since they are made to mimic human brains, traditional swarming wouldn't work but it would be possible to mimic collaborative intelligence between these machines. These group decisions have the possibility of being more "human" due to the aggregate of the minimal bias within each machine, being more accurate, etc.

This project explores the results of using an optimized group-work algorithm made for use with human responses in different chatbots to see if the bots are able to generate more "human" responses when they work together compared to when they work separately.

Background and Related Work

1. Swarm algorithms

First, biologically inspired algorithms were observed to see how current works applied swarm intelligence to computational problems, such as the Traveling Salesman Problem. Most specifically, I investigated works concerning the implementation of Ant Colony Optimization (ACO), which is a metaheuristic based on the behaviors of the ant colony to solve combinatorial optimization problems. The way that ant colonies are able to solve for the shortest path to a certain food source is based on the use of pheromones and the principle of stigmergy, as defined above. When an ant finds a food source it lays out a pheromone trail that is followed by subsequent members of the colony with a certain probability. Because the strength of these pheromone trails decreases over time, the shortest paths are laid out with stronger pheromones and are more likely to be chosen by the members of the colony. ***Figure 1*** illustrates this process by splitting the procedure into 3 steps: exploration, optimization, and research extraction. The exploration step represents the findings of the first few ants, the optimization step illustrates the process by which the entire colony works to optimize the path to the food, and the research

extraction illustrates the results. The ACO implements this behavior by introducing artificial ants that have the ability to produce simulated pheromone trails onto a "*construction graph*" representing a specific problem [13].

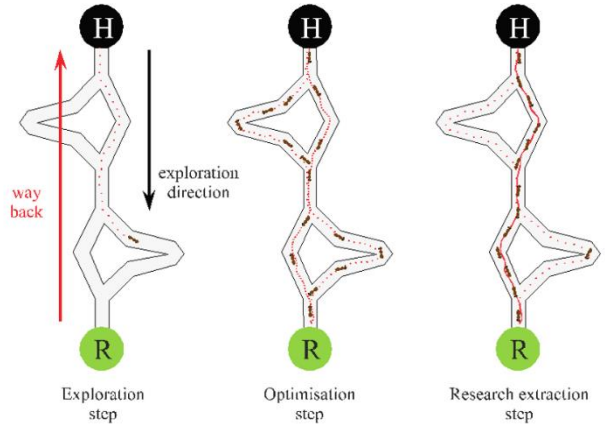

Figure 1: *Illustration of the ant colony optimization process created by Géza Katona [9]*

2. Swarm Robotics

To study how swarming could be applied to higher intelligence, current works surrounding multi-agent swarm systems, a field which attempts to mimic the swarming abilities of organisms such as bees and ants to develop a decentralized system of robots with the ability to cooperate, were looked at. In this field, rather than being a collection of agents, a swarm is a type of "quasi-organism" [14] where the individuals are as simple as possible while still retaining the ability to sense its local environment and cooperate with its neighbors to produce certain behaviors such as spatial organization, navigation, and consensus. These behaviors require a balance between exploitation and exploration that can be modeled using stochastic components, such as randomized motion, along with heuristic components, such as predefined constants to govern attraction and repulsion [8].

I was most interested in consensus behaviors and in my search found the following work: Cooperative sensing and recognition by a swarm of mobile robots, which was generated by the Dalle Molle Institute for Artificial intelligence [4]. The paper investigated the ability to utilize swarming to increase the accuracy of classification systems, with a good amount of success. They took advantage of the multiple viewpoints and mobility of swarm robots and equipped each with a statistical classifier so it could generate an individual opinion. These opinions are then distributed throughout the network and a decision is reached using a specific consensus protocol. In this paper they also defined the conditions necessary to reach a swarm-level decision: an evidence trigger or a time trigger. The evidence trigger works by each robot immediately adopting an incoming decision if it is supported with more evidence than its current decision. Alongside this, the time trigger serves as a sort of countdown for the systems to ensure quick decisions.

3. Unanimous.AI

. Swarm.ai is a project that attempts to optimize group-work amongst "human-swarms" by using the similarity between the principles of swarming and leaky integrations in neural populations. Both principles are based on the premise of parallel "excitable-units" that are able to cooperatively, "integrate noisy information, weigh competing alternatives and converge on optimized solutions in synchrony" [1], with convergence happening once a certain decision threshold (a similar concept to the triggers in the work done by the Dalle Molle Institute) as illustrated in ***Figure 2*** . In order to extend this model to be of use to humans two invariants had to be maintained: "1. support signals must be continually maintained over the decision period or lose influence and 2. decisions are reached when a threshold level of activation is exceeded" [1]. In order to maintain these the unanimous.ai team used a high-level physics model to

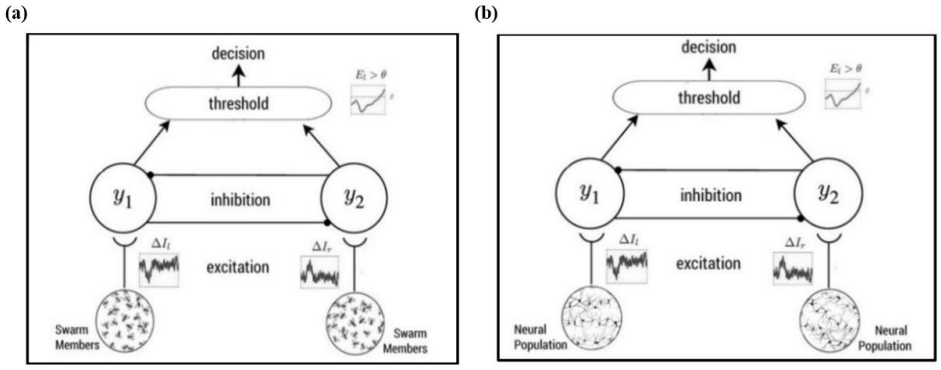

Figure 2: *Figures illustrating decision making created by unanimous.ai [1]*
(a) *Usher-McClelland model of neurological decision-making* ***(b)*** *Mutually inhibitory decision model in bee swarms*

represent the deliberation process. In the swarm.ai systems, each person (or "excitable unit") in the network is given a magnet as an avatar white and the decision was determined by a graphical puck that would be under the influence of all of the magnets. The graphical puck is moved by the forces exerted by each of these magnets and a decision is reached when the puck's distance from a certain decision point is low enough, thereby fulfilling the second invariant described above. The user is able to manipulate the magnet towards the outcome they prefer and the strength of this influence is determined by how close the magnet is to the puck. The puck is in constant motion so unless the user exerts a constant force in the form of manipulation by a mouse or

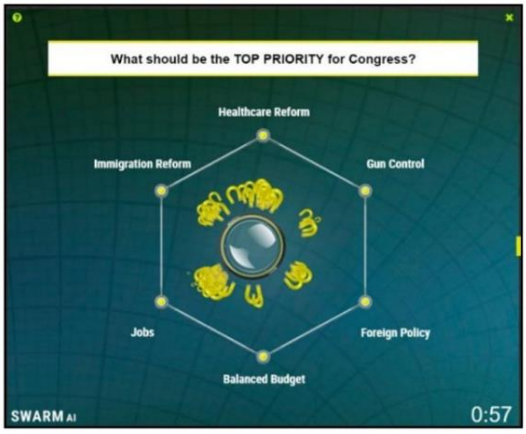

Figure 3: *Example of human swarming process created by unanimous.ai[1]*

touchscreen the amount of force they are able to communicate to the puck will be minimal. This system ensures that the signal provided by a certain user will die out over time unless it is continuously supported, which fulfills the first invariant stated above. ***Figure 3.*** shows an example of the human swarm system.

Approach

A swarm intelligence approach was used for the simulation of group work between different chatbots to develop an emergent intelligence that would be able to use the subtle biases within each program to generate more insightful answers. Traditional swarming methods, mainly the ACO and swarm robotics technologies, were, however not the avenue pursued because they did not take proper advantage of the superior intelligence of systems such as chatbots. Both these methods, instead, focused on maximizing the possible results that could be achieved while using the simplest individual agents possible.

Swarm.ai, on the other hand, placed a lot of emphasis on the complexity of their individual agents, portraying them as humans. Traditional avenues of group work amongst people, such as surveys and polls, fail to consider variables such as the certainty of the individual or their amount of concern with the issue in general. By forcing the user to maintain a continuous motion with their mouse, swarm.ai can compute more thorough solutions by taking in a large vector from each individual, rather than a simple yes or no response.

Most modern chatbots use an artificial neural network in which multiple layers of nodes, similar to neurons in the brain, are connected non-linearly in order to build their responses in order to mimic how humans build responses. As such the similarities between the chatbot neural network and the human neural network suggest that methods of group work that are optimal for humans could be optimal for bots as well.

The main approach of this project is using a physics model for the base, similar to swarm.ai's, where bots are the main users that have the abilities to control magnets that influence a decision puck towards a certain decision point. The bots will each be computationally attributed factors such as doubt and concern based on their performance which will determine the force which they are able to apply to said magnet. Success was defined as the model passing 3 types of evaluations: timing, accuracy, and similarity to human responses.

Implementation

1. How the bots will affect their magnets

In order to determine the magnet's movements each bot has the ability to determine the magnitude and direction of a net force that will push the graphical magnets. Their ability to do so is represented as the *update_F()* function within the *ChatbotPuck* data structure, which was coded in python. A simplified API of this data structure is shown in ***Figure 4.***

```python
# Author Lois I. Omotara
class ChatbotPuck():
    def __init__(self,name, convic_words=None, convic_freq=None, model=None,
    debug = False):
        # initializes data structure the involves loading the responses and
        # frequences of the bot
        pass
    def choose(self,convic_words,convic_freq):
        # uses responses and frequencies of the bot to choose a decision point
        pass
    def change_choice(self):
        # when the bot changes it choice this function updates the position
        # of the bot to match the initial for that decision point and
        # updates the direction of the force as well
        pass
    def update_sim(self,word = None):
        # updates/creates similarity vector by using SentenceTransformer
        pass
    def update_F(self):
        # updates the force by using the conviction vector
        # also deals with the possibility of changing answers and, in that
    case
        # dampens the force accordingly
        pass
    ...
    def update(self):
        # updates puck at each time step
        pass
```

Figure 4: *Simplified ChatbotPuck API (full can be found in the* ***Appendix****)*

1.1 How the bot's determine the magnitude of the force applied on the magnet

In order for the model to represent an emergent intelligence, the individual agents are required to be continuously engaged with the decision-making process or risk their support signal waning over time. To fulfill this requirement, the bot's support for a certain answer must be quantified. This was done using a piece of data called the conviction vector. Like all machinery, chatbots are not always consistent in their responses. Depending on the type of prompt and the number of times the bot is run, it is possible for there to be a large variation in the responses of the bot, a phenomenon that can be seen as parallel to the human emotion of doubt. To represent this, the bot is run with the same prompt multiple times and the responses, and their frequencies are saved into two vectors. s. The vector containing these frequencies are run through the SoftMax function provided in the SciPy package and is transformed into a probability distribution with the largest probability being assigned to the highest frequency. The magnitude of the force on each bot's magnitude is initially set to 1, then, to simulate the continuous engagement, each force is dampened by a factor of k= 0.8, with a probability equal to the value 1-cp with cp=current conviction. Therefore, a bot with less conviction is more likely to have its force dampened than a bot with more conviction.

1.2 How the bots choose and move towards their answer choices

Each bot is prompted with a query and returns a response that will be fed as the input to the model. As stated earlier, the bot is run multiple times and their dedication to each response is represented in the conviction vector. The word that has the highest conviction is represented as an instance variable called *self.word.* However, it is not guaranteed that this value will be available as an answer choice (otherwise known as a decision point) in the model, as such, it is necessary to develop a mechanism for scoring the similarity between the bot's response and the

possible decision points. That is done using a SentenceTransformer package [12]. This package is a transformer architecture that takes in words or phrases as inputs and transforms them into word embeddings which are high dimensional vectors that have the ability to describe the contextual meanings of a word or phrase. These embeddings also have the property that words or phrases that are similar will also be close in that vector space, as such, one is able to score the similarity of two words using the cosine similarity score between their two embeddings. One example of such a vector space is shown in ***Figure 4***. This is used in this project to create a similarity vector in the function *update_sim()* in the *ChatbotPuck* data structure, which holds the similarity score between the bot's output and each of the decision points. The decision points whose similarity score is the highest is chosen as the answer for that particular bot. In the case that two responses have the same value in the conviction vector, the one that has the highest maximum similarity score will be chosen and the decision point associated with that similarity score will be designated as the choice for that bot. Once the choice has been made, the magnet will activate the function *change_choice()* which automatically updates the position of the

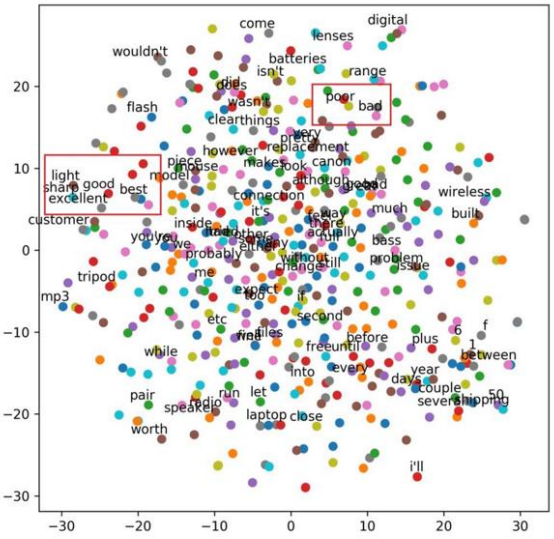

Figure 4: *Illustration of word embedding vector space generated for the publication "Cross-domain sentiment aware word embeddings for review sentiment analysis" [5]*

magnet to the initial position that is associated with that particular decision point and the direction of the magnet.

1.3 Representing Uncertainty

It is also possible that a bot can switch to another answer if it is not extremely confident in its current one. This is represented by an if statement with two conditions. The first condition is that a randomly generated value between 0 and 1, created using the function *random.random()* , is greater than the current conviction + 0.1, meaning that the loop will be entered with the probability 1-cp+0.1. This condition ensures that the loop can only be entered if the current conviction is relatively low and the 0.1 factor is added in order to ensure the loop isn't entered too often. The second condition is related to the previously defined similarity vector. If the word that the bot initially inputs is not an answer choice and is relatively similar to all the answer choices provided, the bot will not care much about the final result and will be inclined to switch more often. As such, in order to enter the loop, the range of the similarity vector must be less than 0.7. In this case a new choice is selected by choosing from the bot's initial responses using the conviction vector as a probability distribution using the function *stats.rv_discrete* and is stored as *self.word*. The similarity matrix is then updated using the function *update_sim()* and the decision point with the highest similarity score is chosen. Then the position and direction of the magnet is altered using the function *change_choice()* to reflect this change. Doing this, however, demonstrates that the bot is not particularly drawn to a single choice, and so, the force is dampened by a factor of 0.6, rather than the regular 0.8, as a sort of penalty.

2. Defining how force is exerted on the puck.

The next step is to determine the effect that each of the magnets will have on the puck. Each magnet will exert a force on the puck that would draw the puck towards its decision point

directly. This force was the same as the one defined earlier that is used by the bot to control the movement of the magnet. The net force on the puck is calculated by summing all the individual forces emitted by all of the bot's magnets and dividing it by the factor of the distance between the puck and the magnets in the *update_F()* function within the *ModelPuck* data structure that defines the behavior of the main puck. The simplified API for this data structure is shown in ***Figure 5.*** As such there is also an infinitive for each of the magnets to stay relatively close to the puck. To represent this there is an instance variable within the *ChatbotPuck* structure known as *self.flip*. Once the distance between the magnet and the puck has reached a certain threshold, the direction of the force that the bot is using to control the magnet will flip, causing it to move closer to the puck. This does not affect the direction of the force from the magnet that is affecting the puck. Instead, that force will always be pulling it towards its chosen decision point.

```
# Author: Lois I Omotara
class ModelPuck :
    def __init__(self,chatbots=None, choices = None,debug=True):
        # initializes data structure
        pass
    def calc_choicesd(self):
        # calculates distance between model puck and all decision
        # points
        pass
    def set_chatbots(self,chatbots):
        # set the chatbots to be used in model
        pass
    def set_choices(self,choices):
        # sets decision points to be used in model
        pass
    def get_choicep(self, choice):
        # get position of each decision point
        passs
    def update_F(self):
        # updates the force influencing the model puck by
        # aggregating the force of all the magnets
        pass
    ...
    def update(self,start):
        # updates model puck at each timestep
        pass
```

Figure 5: *Simplified ModelPuck API (full can be found in the* ***Appendix****)*

3. Creating the High-Level Physics model

Once all the force calculations were complete, the final simulation was generated using the n-body simulation as a template. The force was updated at each time step based on the calculations above The acceleration, velocity, and position were updated using the following calculations (except for when a choice change was made at which the position would be updated according to the rules stated above):

$$a_t = \frac{F_t}{m}$$

$$v_{\{t+1\}} = v_t + a_t * dt$$

$$p_{\{t+1\}} = p_t + v_t * dt$$

Restrictions were placed on the positions of the magnets and the puck so that none of the objects could move outside of the designated board. Moreover, once the position of the puck was sufficiently close to one of the decision points (which would be determined by the value of the distance threshold) the model would converge on that decision. The animation was made using the *FuncAnimation* function from the matplotlib.animation package and a snippet of it is shown in ***Figure 6.***

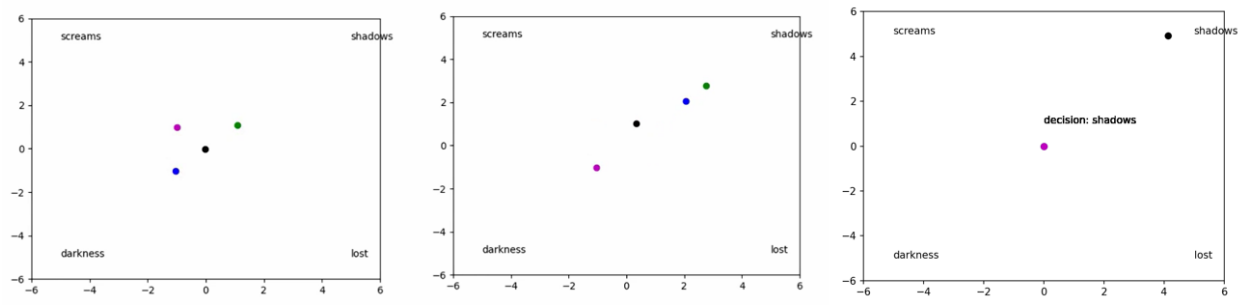

Figure 6: *Sample progression of the animation of the model. Each of the colored dots represent the magnets controlled by the bots while the model puck is shown in black.*

Evaluation

1. **Chatbots**

The chatbots used for the evaluation portion of the project include the following: chatgpt, claud.ai, koalachat, perplexity.ai, hugging chat (using the mistralai /Mixtral-8x7B -Instruct-v0.1 model) , and pi

2. **Timing**

The first evaluation metric tested the system on the amount of time it took the system to converge on an answer choice. This was evaluated using a python timing package. All tests in this section of the evaluation were done with the prompt: “What’s the best word to begin a horror story”. Moreover, to see how fast the model converged by itself, all tests below were run without the animation code.

To begin a simple round of timing tests were conducted where I used three bots (chatgpt,claud.ai, and koalabot) that were run with the prompt, defined above, 10 times each and 4 decision points (“darkness”,”shadows”,”screams”,”lost”) that were chosen randomly from all the words that the bots had generated. Since this was a simple prompt with few bots ,success was defined as the model converging in under 10 seconds. After running the model 100 times, without the animation code, it was found that on average it converged on the word “shadows” after 5.1730 seconds.

The next round tested how swarm density affected the timing of the model’s convergence. r A model was run with the procedure described above using 2,3,4,5, and 6 different bot’s. usage of the bots was done in the order described in the **Chatbot** section. The results are summarized in ***Figure 7*** and ***Table 1***. The model only passed this evaluation when 4 or less bots were used. It can also be seen that convergence time increases linearly with the

addition of bots. In the following sections the model was run repeatedly for 30 minutes to determine the number of trials.

Amount of Bots	Time	Decision
2	2.515175569	shadows
3	5.17298712	shadows
4	7.582761054	31% darkness 68% shadows
5	10.52215667	43% darkness 56% shadows 1% screams
6	11.55337122	darkness

Table 1: *Table showing results of timing evaluation based on swarm density*

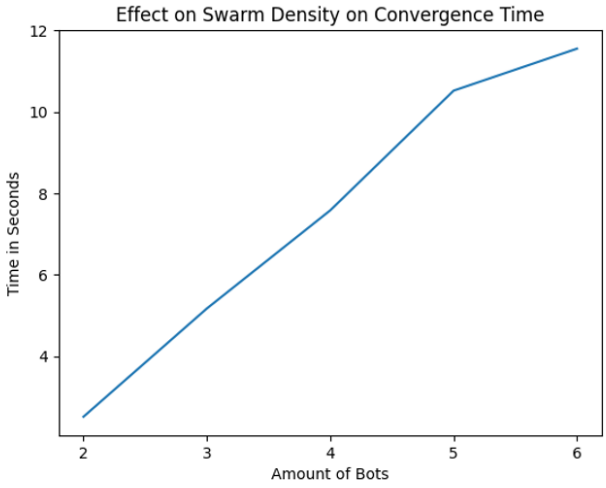

Figure 7: *Chart showing how the time it takes for the model to converge is related to the swarm density*

3. Accuracy (Simple Answer Choices)

The next evaluation metric tested was the accuracy of the model at gravitating towards the answer choice that garnered the most support from the most amount of bots. To do this, a model was configured for use with 4 bots and 4 decision points, and it was loaded with dummy responses and conviction frequencies. The behavior of the model was then analyzed.

The first round of testing used the dummy responses “one”, “two”, “three”, and “four” which also were used as the decision points. The conviction frequencies for each bot were as follows: [8,1,1,0] , [2,3,3,2], [1,1,1,1]. Since bot1’s conviction for the answer “one” is far greater than the other bot’s conviction in their answers, success for this round would be the model consistently converging on the answer “one”. Out of the 95 trials run, 91 returned the decision “one” and the other 4 returned the decision of “three”. Since “one” is the clear majority, the model passed this evaluation.

The second round of testing analyzed the behavior of the model when there was a “tie” in the convictions of some bots. Overall, it was necessary to make sure one of the top decision points were chosen rather than a random third party. Using the same dummy responses and decision points as in the previous round, the new conviction frequencies were: [1,1,1,7], [1,7,1,1], and [1,1,1,1]. Success for this round was defined as the decision being “two” or “four” over 70% of the time. Out of the 68 trials run, “two” was chosen 27 times and “four” was chosen 41 times, which can be defined as a success.

4. Accuracy (Complex Answer Choices)

This section of testing evaluated if the accuracy of the model held when the decision points were different from the straight-forward responses of the bots in order to test the efficacy of the similarity vector mechanism.

For the first round of testing used the decision points: “good”, ”bad” ,”sheep” , and ”yellow”, the responses: [“great”, “like”, “red”, “blue”], [“pink”, “goat”, “heart”, “wallet”], [“mattress”, “door”, “evil”, “car”] and the corresponding frequencies were as follows: [8,1,1,0] , [2,3,3,2], [1,1,1,1]. Success for this round was defined as the chatbot deciding on “good” 70% of the time because the phrase with the highest frequency is “great” which is very similar to “good”. Out of 100 trials “good” was chosen 100 times so the model passed this evaluation.

For the second round of testing the same decision points and the response vectors were kept, but the frequencies were changed to be: [1,1,1,1] , [1,7,1,1], [1,1,7,1]. Success for this round was defined as the model picking “bad” or “sheep” since the words with the greatest corresponding frequencies were “goat” and “evil”. Out of 13 trials “bad” was chosen 8 times and “sheep” was chosen 5 times so the model passed this evaluation as well. It is also interesting to note that, despite running for the same amount of time as the previous tests, the model was only able to converge 13 times, a significant decrease.

5. Similarity to human responses

The last evaluation tested if the responses generated by the group work represented in the model was more accurate when it came to open-ended questions. Since these prompts are more open-ended, accuracy was defined as the answers most people would have chosen. To represent this, a dataset generated from family feud poll results and the top four top answers were used as decision points[11]. There were two definitions of success for this evaluation: 1. The model chose the top answer and 2. The model’s answer was more correct, or higher in the rankings, than at least one of the chatbots used . The three chatbots used in this section were cgpt, claud, and koalachat. Out of the 8782 prompts available in the dataset, 5 prompts were chosen and ran for 20 trials each. Similar to the procedure in the previous sections, each chatbot was run with the prompt 10 times and their responses along with the frequency of each response was used to load the model.

Prompt 1: ‘Name An Occupation Where People Are Paid To Tell Others What To Do (More Specific Than Boss)’

The top four answers for this prompt were “doctor”, “lawyer”, “fitness trainer”, and “account broker”, in order from most to least popular. All three bot’s chose the answer “doctor”

and the model converged on this as well. Since the model converged on the most popular answer it passed this round of the evaluation.

Prompt 2: 'Name A Reason Why It's Harder To Get Out Of Bed On Some Days Than On Others. '

The top four answers for this prompt were "tired", "sick", "bad weather", and "going to work/school". All the bots and the model converged on the top answer in all 20 trials so the model passed this round as well.

Prompt 3: 'Name a kind of place that's known for serving really bad food. '

The top four answers for this prompt were "cheap restaurants", "hospital" , "school", and "convenience stores". The cgpt bot chose the answer "school", the claud bot chose "cheap restaurant", and the koalachat bot chose "hospital". Out of the 20 trials the model converged on "hospital" 10 times and "school" 10 times. The model passed this round of this evaluation in 10 trials because it was able to choose "hospital" which is higher ranking than cgpt's answer of "school".

Prompt 4: 'Name A Food That Can be Eaten Directly From Its Container.'

The top four answers for this prompt were "ice cream", "yogurt", "chips", and "tuna". All the bots and the model chose" yogurt "in all of the trials. The model did not pass this round of the evaluation because it did not perform better than any of the bots.

Prompt 5: 'Name A Fruit Put in Margaritas'

The top four answers of this prompt were "lime", "strawberries", "lemon", and "pineapple". All the bots and the model chose "lime" in all the trials. The model passed this round of the evaluation because it was able to converge on the most popular answer.

Overall, since the model was able to choose the top answer for 3 of the prompts out of the 5 it passed for the first definition of success. However, since out of the 40 trials where the model did not converge on the top answer, it was only able to outperform at least one bot in 10 trials, it did not pass for the second definition of success.

Conclusion and Future Work

This project focused on the possible benefits of pursuing swarming amongst artificial intelligences by simulating group work amongst different chatbots. The model that was created to this end was a physics-based model that allowed the chatbots to interact based on factors such as their conviction in a certain answer. In the model each chatbot was given a magnet with the ability to influence a decision puck towards a decision point representing a certain answer. The model was found to be accurate, in terms of choosing answers that corresponded to the highest conviction, with simple and complex answer choices. Moreover, when using 4 or less chatbots it passed the timing criteria by converging in under 10 seconds on average. Through testing the model was able to accurately and relatively quickly converge on answers that were most like human responses 60% of the time. In the trials where the model was unable to choose the most human answers, the model outperformed at least one of the chatbots 25% of the time. Though these results did not meet the criteria for success, it does show possibility for better results should further experimentation occur with variables such as swarm density and non-linear motion of the bolt's magnets. Exploration of this potential could lead to the development of AI "groups" that are able to make thoughtful decisions without excessive human intervention. Further work could involve exploring using large amounts of answer choices or even the efficient generation of answer choices. Moreover, it could be worthwhile to explore the use of

nonlinear motion in the model or connecting multiple models in a system that would allow for the generation of large amounts of text.

Acknowledgements

I would like to acknowledge my advisor, Bernard Chazelle, for all the great feedback he's given me this semester and Gregg Wilcox, a partner from unanimous.ai who was especially helpful.

Honor Code

I swear on my honor that the above work is my own.

- Lois I. Omotara

Bibliography

[1] L. Rosenberg and G. Willcox, Artificial Swarm Intelligence. 2019.

[2] “ChatGPT.” Accessed: Jan. 11, 2024. [Online]. Available: https://chat.openai.com

[3] “Claude.” Accessed: Jan. 11, 2024. [Online]. Available: https://claude.ai/chats

[4] A. Giusti, J. Nagi, L. Gambardella, and G. A. Di Caro, “Cooperative sensing and recognition by a swarm of mobile robots,” in 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, Vilamoura-Algarve, Portugal: IEEE, Oct. 2012, pp. 551–558. doi: 10.1109/IROS.2012.6385982.

[5] J. Liu, S. Zheng, G. Xu, and M. Lin, “Cross-domain sentiment aware word embeddings for review sentiment analysis,” International Journal of Machine Learning and Cybernetics, vol. 12, Feb. 2021, doi: 10.1007/s13042-020-01175-7.

[6] “HuggingChat.” Accessed: Jan. 11, 2024. [Online]. Available: https://huggingface.co/chat

[7] “Koala - The Best AI Writer and Chatbot.” Accessed: Jan. 11, 2024. [Online]. Available: https://koala.sh/

[8] H. Hamann and T. Schmickl, “Modelling the swarm: Analysing biological and engineered swarm systems,” Mathematical and Computer Modelling of Dynamical Systems, vol. 18, no. 1, pp. 1–12, Feb. 2012, doi: 10.1080/13873954.2011.601426.

[9] G. Katona, B. Lénárt, and J. Juhasz, “Parallel Ant Colony Algorithm for Shortest Path Problem,” Periodica Polytechnica Civil Engineering, vol. 63, Jan. 2019, doi: 10.3311/PPci.12813.

[10] “Pi, your personal AI.” Accessed: Jan. 11, 2024. [Online]. Available: https://pi.ai/talk

[11] “protoqa-data/data at master · iesl/protoqa-data,” GitHub. Accessed: Jan. 11, 2024. [Online]. Available: https://github.com/iesl/protoqa-data/tree/master/data

[12] “sentence-transformers (Sentence Transformers).” Accessed: Jan. 11, 2024. [Online]. Available: https://huggingface.co/sentence-transformers

[13] D. O. and S. A., “Swarm Intelligence from Natural to Artificial Systems: Ant Colony Optimization,” J GRAPH-HOC, vol. 8, no. 1, pp. 9–17, Mar. 2016, doi: 10.5121/jgraphoc.2016.8102.

[14] M. Schranz, M. Umlauft, M. Sende, and W. Elmenreich, “Swarm Robotic Behaviors and Current Applications,” Frontiers in Robotics and AI, vol. 7, 2020, Accessed: Jan. 11, 2024. [Online]. Available: https://www.frontiersin.org/articles/10.3389/frobt.2020.00036

[15] G. Willcox, “Swarm.ai,” Nov. 03, 2023.

[16] “What is Sentence Similarity? - Hugging Face.” Accessed: Jan. 11, 2024. [Online]. Available: https://huggingface.co/tasks/sentence-similarity

Appendix

Github link: https://github.com/lo2173/COSIW

ChatbotPuck full API:

```python
# Author Lois I. Omotara
class ChatbotPuck():
    def __init__(self,name, convic_words=None, convic_freq=None, model=None, debug = False):
        # initializes data structure the involves loading the responses and
        # frequences of the bot
        pass
    def choose(self,convic_words,convic_freq):
        # uses responses and frequencies of the bot to choose a decision point
        pass
    def change_choice(self):
        # when the bot changes it choice this function updates the position
        # of the bot to match the initial for that decision point and
        # updates the direction of the force as well
        pass
    def get_p(self):
        # returns current position
        pass
    def get_v(self):
        # returns current velocity
        pass
    def get_a(self):
        # return current acceleration
        pass
    def update_sim(self,word = None):
        # updates/creates similarity vector by using SentenceTransformer
        pass
    def update_F(self):
        # updates the force by using the conviction vector
        # also deals with the possibility of changing answers and, in that case
        # dampens the force accordingly
        pass
    def update_a(self):
        # updates acceleration
        pass
    def update_v(self):
        # updates velocity
        pass
    def update_p(self):
        # updates position and keeps magnet from going passed borders
        pass
    def reset(self):
        # resets instance variables
        pass
    def update(self):
        # updates puck at each time step
        pass
```

ModelPuck full API:

```python
# Author: Lois I Omotara
class ModelPuck :
    def __init__(self,chatbots=None, choices = None,debug=True):
        # initializes data structure
        pass
    def calc_choicesd(self):
        # calculates distance between model puck and all decision
        # points
        pass
    def set_chatbots(self,chatbots):
        # set the chatbots to be used in model
        pass
    def set_choices(self,choices):
        # sets decision points to be used in model
        pass
    def get_choicep(self, choice):
        # get position of each decision point
        pass
    def get_dt(self):
        # get dt being used in model
        pass
    def get_p(self):
        # get model puck's position
        pass
    def get_v(self):
        # get model puck's velocity
        pass
    def get_a(self):
        # get model puck's acceleration
        pass
    def update_F(self):
        # updates the force influencing the model puck by
        # aggregating the force of all the magnets
        pass
    def update_a(self):
        # updates acceleration of model puck
        pass
    def update_v(self):
        # updates velocity of model puck
        pass
    def update_p(self):
        # updates position of model puck
        pass
    def update(self,start):
        # updates model puck at each timestep
        pass
```