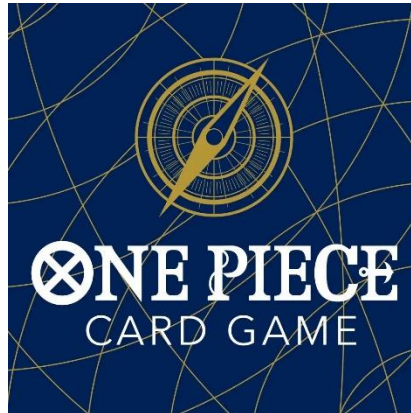


ONE PIECE TCG COMPANION CHATBOT



Andrew Gerungan

https://github.com/Djeggis/optcg_companion/

0 NLP CONTRIBUTIONS



Andrew Gerungan – data scraping, deck statistics, openai implementation

1 CONCEPT

One of the biggest obstacles to approaching any trading card game from *Magic the Gathering* to *Pokémon Trading Card Game* is the abundance of information you need to process both in and out of game. As such, I set out to build an interface to help alleviate the out of game processing of information, mostly found in deckbuilding and rules comprehension. The result of this endeavor was the One Piece TCG Companion Chatbot (from here referred to as OPTCG Chatbot), which currently serves as a medium for analyzing user inputted decks to give an informed decision on how to improve them based off of successful winning decklists across the world. The scope of the project was as follows: a chatbot with a discord interface that allowed users to input decklists and receive suggestions for the One Piece Trading Card Game (OPTCG).

2 DATA COLLECTION

The first hurdle in creating the OPTCG Chatbot was gathering the dataset of winning decklists – there is no readily available formatted dataset for OPTCG and as such, I first began by building a data scraping program. I began by identifying possible websites to scrape data from, eventually settling on <https://onepiecetopdecks.com/>, both based off the abundance of data, formatting of data, and ease of permission. The data scraping program was built in python and utilized the selenium package, creating a proxy Firefox instance that could navigate and interact with the website while scraping deck data directly from the HTML of the page. The biggest problem came in how the website formatted its decklists – the table of decklists did not change the address of the page but was rather an interactable js table. The solution was to ensure that the data scraping program would scroll to the bottom of the page after all elements were loaded and then interact with the table page button, ensuring no data was missed and that the button was actionable. Another interesting issue that arose was that the website sometimes required ads to load the decklist pages. This was circumvented by loading a custom Firefox session with the adblocker extension preinstalled. From there, the data was formatted into a csv file to be processed for later.

	st10luffy	11/3/2024	Indonesia	Ignatius Ivan	1st Place	Indo BCF(10-2)	Bandai Maxsoft
	st13luffy	11/3/2024	Indonesia	Iki Azu	Top-8	Indo BCF(8-2)	Bandai Maxsoft(238)

Showing 1 to 50 of 309 entries

◀ 1 2 3 4 5 6 7 ▶

Figure 1: Navigation interface of <https://onepiecetopdecks.com/>

3 DATA PROCESSING

After data collection, workflow was split into two phases: data processing and chatbot implementation. I began working on the data processing, determining how best to compare a given decklist against the sea of winning deck recipes. I settled on two ideas – comparing average card quantities and turning each card into a vector akin to word2vec. I was originally

going to implement both, but a problem arose with turning each card into a vector – while card relationships are useful in determining if a card can be linked to a certain leader card, they don't actually give a good indication of related non-leader cards, as it fails to consider what leader the deck is for, the color combination, and quantity ratios. As such, I moved forwards with comparing average card quantities. This was accomplished by turning the dataset into an xarray, where the coordinates were the leader card and a specific non-leader card and the value was the average quantity found across winning decklists. This would then be compared against a user inputted decklist to find where changes could be made, sorted by how large of a difference between the user's list and the data's list was.

```
['0: +NEW copies, average inclusion quant: 1.0\n', 'OP07-047: +NEW copies, average inclusion quant: 3.0714285714285716\n', 'OP08-047: +NEW copies, average inclusion quant: 2.857142857142857\n', '04: -copies, your 4 vs data 3.0\n', 'OP06-047: -copies, your 3 vs data 2.7142857142857144\n', 'OP07-048: -copies, your 4 vs data 2.7857142857142856\n', 'OP01-077: +NEW copies, average inclusion q\n1.4285714285714286\n', 'ST03-005: -copies, your 4 vs data 2.357142857142857\n', 'OP02-054: -copies, your 4 vs data 0.5\n']
```

Figure 2: Raw data of comparing average card quantities against user decklist

4 CHATBOT & INTERFACE

The next phase involved implementing the acquired information into a discord chatbot. The first step was to make the skeleton of a discord chatbot utilizing the discord python package – this was relatively simple and did not require very many specific modifications beyond the base outline. Following that, I implemented an openai-4o-mini model – the choice of 4o mini was because the general task is not too complex as well as it's cheaper to use. The next step was then to implement the processed data into the OpenAI system, but I ran into an issue of modifying the chatbot to function how I wanted – the implementations I worked with were inelegant and clunky, and the model would not work with the suggestions I created unless explicitly using a command. While unideal, the time I had to work on the project was dwindling and I settled on maintaining the bot functionality with explicit calls to the suggestion framework I built.

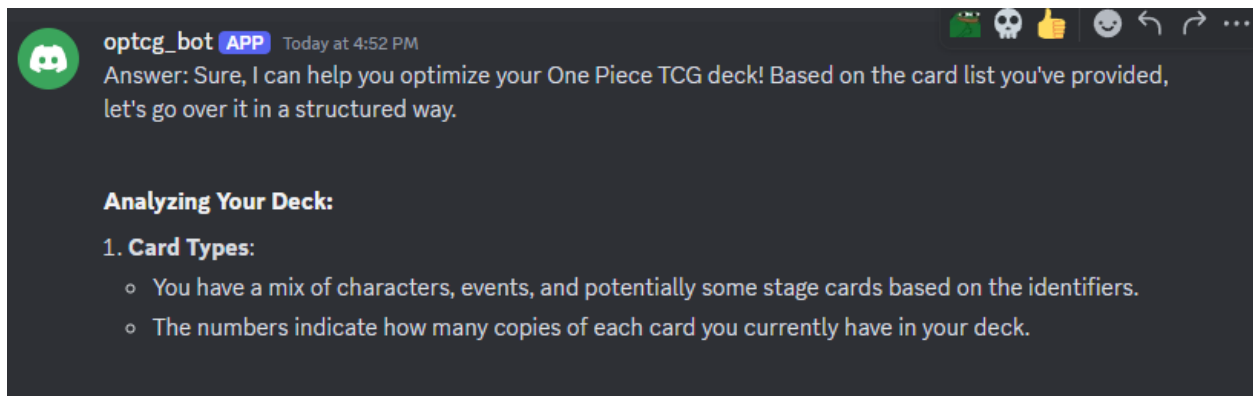


Figure 3: the OPTCG Chatbot working in the Discord environment

5 DISCUSSION

While I was satisfied somewhat with how I was able to create a working interface that utilized both the OpenAI model and my suggestion framework, my inability to blend the two together cleanly was not satisfying. Nevertheless, the experience I gained from working on each phase of the package was immensely helpful and will be of use in the future. Specifically, I feel much more proficient in working with Python and its various packages, as well as tinkering with NLP related areas such as data scraping and LLMs.

I placed the bot into a server to get some secondary opinions as well. These users (6) were mostly familiar with the rules and general game theory of OPTCG but 1 user was only familiar with the basic rules. The main feedback I received was that the suggestions implementation felt clunky, which aligned with how I felt about the inability to seamlessly integrate it into the chatbot. Additionally, I was given recommendations on possible additions, most which I considered and discuss in the following paragraph. Overall, reception was positive but desired more functionality.

Regarding future work, the immediate extension (beyond figuring out how to elegantly implement my deck suggestion) is teaching the model how to convey the rules of the game for both newcomers and veterans alike, most likely coming in the form of processing both the

compact and comprehensive rulings for OPTCG (which is readily available online). Another goal would be to implement the actual cards themselves into the program, rather than simply their IDs, which would entail creating a database for the name, effect, cost, etc. of the card. However, my stretch goal is to work out an implementation to the fan-made OPTCG Simulator, which besides Discord is the place where the use of deck statistics and an assistance-based chatbot would be most useful. The sim itself also has the database of actual cards, so I wouldn't need to construct that myself. To this end, I contacted the creator of the simulator, Maebatsu, and have been given access to the github repo of the project alongside pointers as where to go from here.

ACKNOWLEDGEMENTS

Thank you to <https://onepiecetopdecks.com/> for acting as the source of the data and to Maebatsu for helping with the next steps for the OPTCG Chatbot.

USER TESTING

For ease of use, I have provided a sample decklist to test the program with

1xST13-003

3xOP02-096

2xOP02-121

4xOP04-083

4xOP06-086

4xEB01-056

4xOP06-106

4xOP07-109

4xST13-007

2xST13-008

4xST13-013

4xST13-014

4xST13-015

2xEB01-051

1xOP02-117

4xST13-019