CS 6320 - One Piece TCG Card Generator

Diego Djordjevic, Caleb Hoang

May 8, 2025

0.1 INTRODUCTION

Our project uses Google's Gemini API to generate unique, novel cards for usage in the One Piece Trading Card Game (OPTCG), using the existing rules and cardset as input. The project's backend uses a combination of ChromaDB (a vector database) and SQLite (a traditional database). ChromaDB is used to obtain a set of cards, and SQLite is used to collect the data from the card to be used as training data. Once card data is queried, it is then sent through Google's Gemini API to generate the text for a new, original card.

The frontend program uses the terminal to prompt the user to select a type of card they want to generate, then walks them through the steps to generate a new card via text prompts.

The Github repo can be found at: https://github.com/DiegoDjordjevic/CS6320-OPTCG-Project

0.2 MOTIVATION

The One Piece Trading Card Game (OPTCG) is a card game, based on the *One Piece* anime, that is rapidly gaining popularity. Bandai, the parent company of the game, has opened up 4 new official stores in the U.S. within the last 2 months, making it extremely accessible to American audiences. Additionally, the largest online OPTCG Community has 80 thousand members. Custom card creation is a crucial part of this community, and gameplay with fan-created cards is often just as popular as the main game. This indicates an active demand for custom cards beyond the officially playable cards. With official sets only containing about 100 cards, released only once every 3 months, custom cards keep the community active during the downtime between expansions. Creating a product to speed up the card creation process is a great way to provide value to the community. Although the target was this fan card creation community, the products value has a path to surpass just that. It was not feasible within this semester long project, but image generation and model refinement could also allow this product to be used when creating new official cards.

0.3 EXPLORATION/INNOVATION

0.3.1 Integration of NLP and standard databases

Our solution to this problem utilizes both SQLite and ChromaDB in tandem in order to take advantage of both database formats.

While ChromaDB is extremely robust and capable of finding cards based on vector embeddings, it struggles when trying to retrieve specific traits of a card (for example, searching for a specific card's name, color, or text). Meanwhile, SQLite struggles to find semantic similarities between different cards - it would be both extremely tedious to search through a SQLite database to find all cards which enable card drawing.

Therefore, after trying and failing to find a satisfying way to implement each database individually, we found that working with both was the most effective solution. By using a custom query to search for cards via the Chroma database (i.e. "draw, straw hat, character"), then searching the returned card values in the SQLite database, it is easier to obtain specific values for card names and effects to use as input into Gemini.

This pipeline is extremely easy to modify - by changing embeddings in the ChromaDB database, the cards obtained from a query change slightly. In addition to refining queries, this makes prompt engineering significantly more intuitive and less difficult.

0.4 LESSONS LEARNED

When designing the database for the program, our original idea was to solely use an SQLite database to pull card information from. However, given that neither of us had experience with vector databases, integrating ChromaDB into our project made our project significantly more robust, in addition to further integrating NLP into the final result. This significantly improved the relevance of cards passed in as context when the user had partially specified the details of the card they wanted to generate.

0.5 PRODUCT TESTING

Although we did not actually ship the product for anyone to test on their end, we had players in the local community test it on our machine, as well as rate the output. Diego is part of the local One Piece Trading Card Game, so he was able to collect their feedback on the generated cards. Through testing the product in the local community, we received a lot of feedback that the cards being generated were quite strong, so a possible next step would be creating a heuristic for card strength. As a temporary fix, we implemented a slide for randomness which gave us significant improvements - the "less random cards" felt more balanced. This was a perfect balance - the online community that creates new cards outside of Diego's local community enjoys experimenting with unofficial cards that are significantly stronger than the base game, so the current model perfectly caters to their needs.

0.6 FURTHER IMPROVEMENT

While we did integrate vector databases into our project, it currently uses the default Google GenAI Embeddings. In the future, in order to get more accurate query results from the ChromaDB database, one step we could take is adding our own custom embeddings to each card - for example, such embeddings could be the card's different color/counter/attributes, but also features from their

activation effects such as "card_draw". This should be a task that can be easily automated, and would drastically improve the effectiveness of the vector store.

Additionally, there are many refinements we can do to enhance the generation of cards. As previously mentioned, adding a function to balance the strength of the card would make our program much more practical for creating cards that can be played in conventional games. Adding a functionality to generate a card that works together with previously generated card would provide users with cards that work well together, allowing them to play with the cards right away.

Finally, one of our stretch goals for this project was adding an image generation function - this would allow us to create fully functional cards that could be printed out and physically used in the trading card game. However, time constraints kept us from doing so - in the future, we would like to add this function.

0.7 CONTRIBUTIONS

0.7.1 Diego

- Used BeautifulSoup and Selenium to efficiently scrape card data from the official One Piece Trading Card Game cardlist on their website ensuring the most up to date data is being used.
- Used Langchain to generalize the pipeline allowing users to modify the code to use the model of their choice and create a RAG chain to easily pass in relevant cards as context depending on the card details specified by the user.
- Updated Vector Store to use Google GenAI Embeddings to improve cohesion with Gemini API.
- Generalize User interaction to be easily expandable to new types of cards.
- Manually fine tuned prompt and vector store retriever to improve results.
- Ensured LLM output was properly parsed and printed to the screen in an easily human readable format
- Found players in the One Piece Trading Card Game DFW community to give feedback on the product

0.7.2 Caleb

- Created and managed the SQLite database to load and organize card data pulled from the official OPTCG website. This included creating the database architecture and corresponding diagram.

- Created the ChromaDB database to take queries and output card ids to be used to obtain information from the SQLite database.

Caleb Grading Score:

- 75 pts - Tweaking input into ChromaDB database to create more fitting output to different queries, restructuring SQLite database to allow for one-to-many relationships for color and attributes to be easier to access.
- 10 pts - Complexity in designing the ChromaDB vector store to specifically integrate with the SQLite database and take advantage of both structures.
- 10 pts - Discussion of lessons learned and further improvement to the ChromaDB vector store in order to get more accurate query results

Diego Grading Score:

- 60 pts - Used RAG to improve LLM results by passing in more relevant cards as context. Generalized product to allow one to be able to easily modify specific components without having to rewrite the entire product.
- 10 pts - Scraping data from a poorly structured website that has no back door API to easily download the data. Optimized RAG chain to get the output in the desired format, ensure card originality, and maintain card playability.
- 10 pts - Discussion of lessons learned and further improvement to the ChromaDB vector store in order to get more accurate query results
- 10 pts - Tested product with more than 5 people outside of my team