Building a Collaborative Recommender System for Magic the Gathering

By

Brian DeNichilo, Information Technology Management

A thesis submitted to the Graduate Committee of

Ramapo College of New Jersey in partial fulfillment

of the requirements for the degree of

Master of Science in Data Science

Spring, 2025

Committee Members:

Donovan McFeron, Advisor

Amanda Beecher, Reader

Sourav Dutta, Reader

Acknowledgments

I would like to thank Dr. McFeron for his assistance and willingness to be my advisor for this project.

I would like to thank Dr. Beecher and Dr. Dutta for being my readers for this thesis committee.

Table of Contents

List of Tables

List of Figures

Abstract

The goal of this research is to address the challenge of deck construction in Magic the Gathering's Commander format, a task requiring players to create a deck of 100 cards from a card pool of over 28,000 different cards while also adhering to the color identity constraints of the card chosen to be their commander. The objective is to develop a recommendation system, a tool that uses collaborative filtering to suggest relevant cards to the player based deck construction patterns of the commander community.

The recommender system utilizes Alternating Least Squares (ALS) matrix factorization to identify latent features which capture the relationship between cards in a Commander deck. A model was trained using 220,000 player-created decks scraped from a popular deck building website. The model was tuned by systematically testing various configurations of hyperparameters which include latent factors, regularization values, confidence scaling, and iteration counts to determine the optimal configuration.

A final model was produced using 600 latent factors, regularization of 2.25, alpha of 10, and iteration count of 25. This parameter configuration resulted in an F1 score of 0.33 and MRR of 0.063. Additionally, it had a precision@5 of 0.64 and precision@10 of 0.58 when tested with a seed of 40%, meaning that 64% of the top 5 and 58% of the top 10 recommendations appeared in the test decks.

Introduction

Magic the Gathering (MTG) is a collectable trading card game created by mathematician Richard Garfield and published by Wizards of the Coast in 1993. Despite being over 30 years old, Magic has maintained its position as one of the world's most popular trading card games with a player base of over 40 million people. Magic has many formats which dictate how the game is played, one of which being Commander. Commander has become one of Magic's most popular formats due to its focus on casual play and unique deck-building structure.

Building a Commander deck can be a daunting task for both new and experienced players. This process requires players to create a deck of 100 card deck (one of which being their commander) from a pool of over 28,000 unique cards. The format also imposes additional constraints in that each deck may contain no more than one copy of any non-basic land and all cards must adhere to their chosen commander's color identity. Given these constraints and the large number of cards to pick from, Commander deck construction requires extensive knowledge of the game and its cards. The current deck building process involves users having to look through extensive card lists or manually search for other decks for inspiration. There is a popular recommender system used by the Magic community, however their methods are not public and seem to be based only on card frequency rather than identifying card synergies.

This research intends to develop a recommender system which assists players in this deck construction process. It will provide players with suggested cards based on the player's chosen commander, the commander's color identity, and cards they may or may not already have chosen to be in their deck. To do this, collaborative filtering through Alternating Least Squares

Matrix Factorization is used to identify patterns and card synergies based on historical deck list data using over 200,000 player-created decks.

Chapter 1 - Background

1.1 Introduction

This chapter covers the background of my work, covering the basics of Magic the Gathering and Recommender systems

1.2 Overview of Magic the Gathering and the Commander Format

This section intends to cover the very basics of Magic the Gathering. Magic is a complex game requiring strategic thinking and resource management. While this section will not give a detailed guide on how to play the game, it will provide enough context for someone with zero knowledge of the game to understand the core concepts of my recommender system.

1.2.1 Magic Basics

The main resource in Magic is "mana", which is generated from "lands" - a type of card. Typically, a player may play one land on each of their turns, which can each be "tapped" (indicated by turning the card sideways) once per turn to create mana. There are five basic land types, each of which is associated with a different color, producing one mana of that color. This mana is used as a cost to play the other non-land cards in the player's hand. Figure 1 shows the different land types, the color mana they produce, and the symbol used on cards to indicate the mana type.

Land Type	Color Produced	Mana Symbol
Plains	White	☀
Island	Blue	💧
Swamp	Black	💀
Mountain	Red	🔥
Forest	Green	🌳

Fig. 1 - Lands, Colors, and Mana Symbols

These other non-land cards come in some combination of colors or may be colorless. A card's color is determined by the symbols in its cost shown at the top right of the card. This cost could be represented as a generic cost, meaning any color of mana can be used to play it, or a color symbol, meaning a specific color of mana must be used to play it. These cards have various effects which are used to achieve the primary objective of dealing damage to reduce your opponent's life total to 0 to win the game.

An example of the Magic card ‘Shivan Dragon’ is shown in Figure 2. At the top-right corner of the card, we can see the card's mana cost. The cost to play this card is 4 generic mana, which can be paid with any color mana, plus two additional red mana.

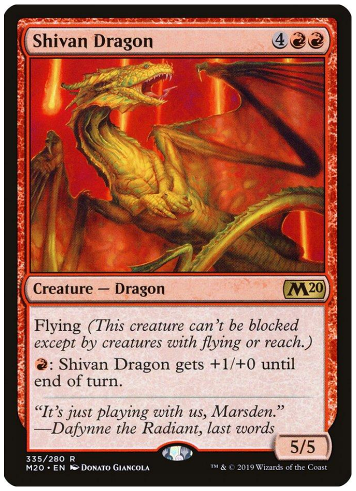

Shivan Dragon
4
Creature — Dragon
M20
Flying (This creature can't be blocked except by creatures with flying or reach.)
: Shivan Dragon gets +1/+0 until end of turn.
"It's just playing with us, Marsden."
—Dafynne the Radiant, last words
5/5
335/280 R
M20 • EN Donato Giancola
™ & © 2019 Wizards of the Coast

Figure 2 - Example of the Magic card *Shivan Dragon*

On the card, directly below the art, we can see the creature's type-line - in this case ‘Creature - Dragon’. ‘Creature’ is the main type and ‘Dragon’ is the subtype. Going forward, when ‘card type’ is mentioned it will be referring specifically to the main type.

Below the type-line, is any rules text for the card which describes what the card does. Each card has a unique effect which creates synergies between cards.

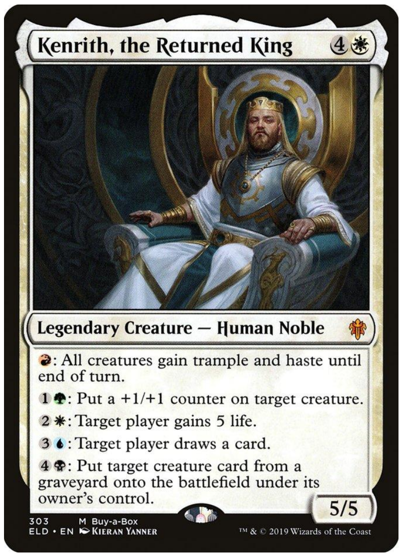
1.2.1 Commander Format

The Commander format differs from a typical game of Magic. It alters how the game is played and the rules for deck construction in the following ways:

1. Gameplay Differences:

    a. Commander is played with 3-5 players (compared to 2 players in a typical non-Commander game)
    b. Players start at a higher life total of 40 (compared to 20)
    c. Each player's commander starts outside the deck and may be played multiple times for an increase cost each time it's played
    d. An alternate lose condition is in place, in which taking 21 or more damage from a specific commander causes you to lose the game
2. Deck Construction:
    a. Each deck must contain exactly 100 cards (99 cards and a commander)
    b. Only one copy of any specific card may be included in your deck (excluding basic lands)
    c. All cards in a players deck must adhere to their chosen commander's color identity

A card's color identity may be different from a card's color. The color identity of a card is determined by the color of the mana symbols in its casting cost in addition to and colors of mana symbols appearing in its rules text. Using a commander with a single color identity would restrict a player's card choices to only using cards of that color and colorless cards (e.g. a red commander would restrict a player to only red and colorless cards). Using a commander with a multi-color color identity would restrict a player to cards which are a subset of those colors and colorless cards.


Kenrith, the Returned King
4
Legendary Creature — Human Noble
: All creatures gain trample and haste until end of turn.
1 : Put a +1/+1 counter on target creature.
2 : Target player gains 5 life.
3 : Target player draws a card.
4 : Put target creature card from a graveyard onto the battlefield under its owner's control.
5/5
303 M Buy-a-Box
ELD • EN KIERAN YANNER
™ & © 2019 Wizards of the Coast
Fig. 3 - Kenrith color identity

In Figure 3, we see a card named 'Kenrith, the Returned King'. This card has a white symbol in its casting cost, meaning this card's color is white. However, the card's rules text contains symbols for each other color, meaning this card's color identity includes all colors. This would mean a person constructing a Commander deck with Kenrith as the commander may add cards of any color combination to their deck.

Additionally, there are restrictions on which cards may be designated as a commander. This commander typically defines the goal and strategy of the deck.

1.3 Recommender Systems

This section gives an overview of the different types of recommender systems. Special attention will be made to collaborative filtering, as it is the implementation chosen for the deck recommender system.

The field of recommender systems encompasses the tools and methods used to provide suggestions, prioritizing items likely to be of interest to the user. It achieves this by learning from a large quantity of data to identify patterns. [1] An example of a Recommender System would be Netflix recommending shows to a user by analyzing their watch history and comparing it to the watch history of similar users.

Recommender systems generally fall under three categories[3]:

- Content-based filtering
- Collaborative filtering
- Hybrid Approach

1.3.1 Content Based Filtering

Content-based Filtering recommends items based on an analysis of their features to match them with user preferences. This method focuses on a single user's behavior and relies heavily on item descriptions and user profiles. We commonly interact with these types of recommender systems when we get suggested news articles or restaurants. The advantage of this method is that it provides personalized recommendations based on the individual's past behavior. A common hurdle to overcome for this type of recommender system is the "new user" problem, in which a new user has little to no behavior history to base recommendations off of.[3]

1.3.2 Collaborative Filtering

Collaborative filtering provides recommendations to a user based on the behavior of multiple similar users.[3] It finds a set of nearest neighbors associated with each user and how they rate each item to predict the preferences of a specific user.[2] Unlike content-based filtering, collaborative filtering does not require explicit knowledge of an item's features to make predictions. Collaborative filtering can be further split into memory-based collaborative filtering or model based collaborative filtering.[3]

Memory-based collaborative filtering uses the entire database to generate recommendations. This method can be further split into two additional subcategories; user-based memory filtering and item-based filtering. User-based memory filtering computes the similarity between users. It identifies similar users and compares their rating on a given item to predict how the target user would rate items they have not rated yet. In contrast, Item-based collaborative filtering computes the similarity between items. It calculates how similar other items are to items the user has already interacted with, then recommends a set of the most similar items.[3]

Model-based collaborative filtering generates a model from the data and learns from past user interactions to improve the performance of the model. A common method to achieve this is through matrix factorization - the method used by my recommender system. The key advantages of model-based collaborative filtering is it's scalability and ability to handle sparse matrices.[3]

My implementation for the Magic Commander deck recommender utilizes model-based collaborative filtering, where 'decks' are treated as 'users' and 'cards' are treated as the 'items'. This is achieved through Alternating Least Squares (ALS) matrix factorization to capture the latent feature relationships between decks and cards, which will be covered in Section 1.4.

A common issue needed to overcome with this type of recommender system is the “cold-start” problem, which is when there is not enough context given to generate meaningful recommendations. This would apply to my recommender system in the scenario where a user provides their chosen commander but no additional cards.

1.3.3 Hybrid Filtering

Hybrid recommenders utilize both content-based filtering and collaborative filtering to provide more optimized recommendations and avoid some limitations of each technique. [3]

1.4 Alternating Least Squares (ALS)

Alternating least squares is the matrix factorization technique used by my deck recommender system. It is a process for taking a large matrix and factoring it into a smaller representation of the original matrix. This section will cover the process of matrix factorization and why it was the chosen technique.

1.4.1 How ALS Works

ALS works by attempting to decompose a large, sparse matrix into two smaller, dense matrices. The original matrix is sparse because there are only 100 cards included in each deck, out of over 28,000 possible cards. These smaller matrices represent the user and item matrices - or in my case, deck and card matrices. These matrices contain the hidden, latent factors which explain why certain cards appear together.[4] Each deck and card is represented by a vector of latent features. The dot product of these vectors predicts the likelihood a card is going to be included in a deck.

ALS solves the matrix factorization problem through the following steps:

1. Start with random values for both matrices
2. Fix the item (card) factors and solve for the user (deck) factors
3. Fix the user factors and solve for item factors
4. Repeat this process until convergence

This can be represented with the equation **R** ≈ **U** x **V**, where **R** is the original deck-card matrix, **U** is the deck-feature matrix, and **V** is the card-feature matrix. The number of features (factors) is a hyperparameter, which must be tuned to find the appropriate value. Figure 4 shows how Alternating Least Squares matrix factorization is applied to my dataset.

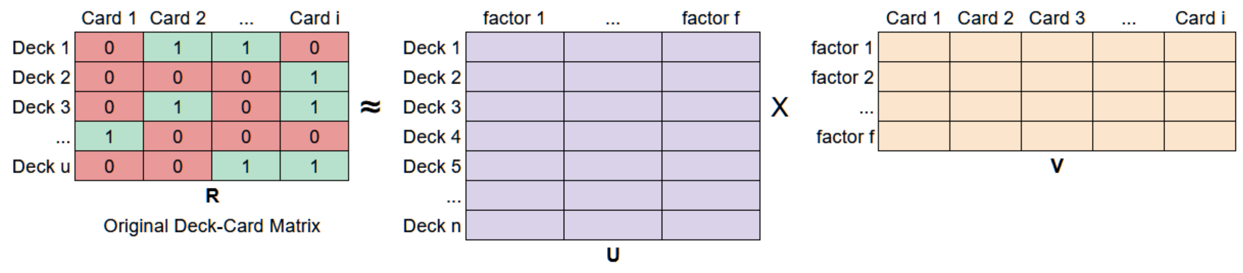


Fig. 4 - Alternating Least Squares matrix factorization

1.4.2 ALS Hyperparameters

The performance of an ALS model can be tuned using the following hyperparameters:

- **Latent Factors:** This parameter controls the number of factors used in the smaller, dense matrices (**U** and **V**) produced by the ALS algorithm. Increasing the number of factors enables the model to capture more complex patterns from the data, however too many factors can lead to overfitting.
- **Regularization:** This parameter helps prevent overfitting by penalizing large values in the factor matrices. Increasing this value provides more of that penalty. Higher regularization produces a more generalized model, whereas lower regularization produces a model which matches the training data more closely.
- **Alpha:** This parameter provides the confidence scaling which increases the importance of implicit feedback. Higher alpha values provide more weight to cards that appear in many decks compared to cards that do not.
- **Iterations:** This parameter controls the number of times the ALS algorithm alternates between fixing the deck factor matrix and card factor matrix. Generally, more iterations lead to better convergence, however there are diminishing returns after a certain point.

1.4.3 ALS with Implicit Feedback

A key feature of ALS is its ability to work with implicit feedback. Unlike explicit feedback, such as rating something on a scale from 1-5 stars, implicit feedback simply indicates whether a user interacted with an item. ALS incorporates this implicit feedback through the tunable confidence parameter, alpha. When solving for deck and card factors, higher alpha values increase the importance of observed positive interactions. [4,5] For my system, the implicit feedback is represented by a card being included in a deck. This is based on the assumption that a card included in a given deck is considered a positive interaction for that deck.

1.5 Conclusion

This chapter provided the background of the Magic the Gathering Commander format, recommendation systems with a focus on collaborative filtering, and Alternating Least Squares which is the method used for matrix factorization. This background provides context for the methodology I used to implement my recommender system described in the next chapter.

Chapter 2 - Methodology

2.1 Introduction

This chapter outlines the methodology used to develop my Magic the Gathering Commander deck recommendation system. It covers data collection and processing, system design, and approach to hyperparameter tuning.

2.2 Data Collection and Processing

The data for my recommender system was over 250,000 player-created decks from the popular deck-building website, Moxfield [6], using their API. This site allows users to build and share their Commander decks. This data was web scraped from between September and December of 2024 and stored into a MySQL database. Additional data about each card, including images of the cards, was obtained from Scryfall [7], a public Magic Card database.

A MySQL relational database was implemented to give the ability to store and query the data efficiently. The primary database entities are:

- **Cards -** This contains the card details, such as name and casting cost, for 28,052 unique Magic cards.
- **Commanders -** The subset of 2,414 cards which can designated as your commander
- **Decks** - 257,103 player-created decks and their commander identities
- **Colors** - Contains color identities for the 5 different colors and colorless cards
- **Types** - Contains the different card types (Creature, Sorcery, etc.)

The tables to manage the many-to-many relationships between these tables are:

- **Deck_cards** - maps the card identities to their associated decks
- **Card_colors** - links the cards with their colors
- **Card_color_identity** - associates the cards with their color identities to ensure adherence to the Commander format restrictions
- **Card_types** - links the cards with their card types

This preprocessed data is what will be used to create the sparse card-deck matrix. The database schema I designed can be seen in Figure 5, which illustrates the entity relationships of the tables in the database.

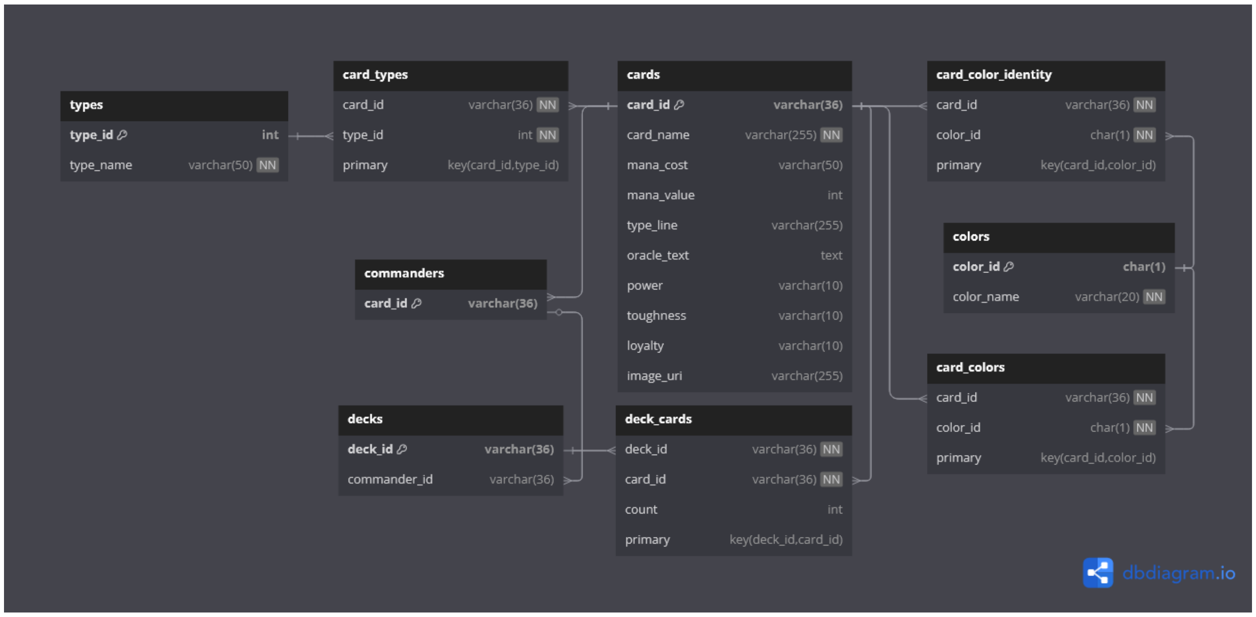


Fig. 5 - MTG Database Design Diagram

2.3 System Design

There are two core components to the recommender system: the MTGDatabase class which handles the database connection and common queries and the ALSRecommenderClass which utilizes the collaborative filtering algorithm. These two components work together to transform the card and deck data into personalized recommendations for the user.

2.3.1 Database Integration

The MTGDatabase class serves as the Data Access Layer, managing database connections and queries. This implementation utilizes SQLAlchemy to connect to the database and Python's built in context manager to ensure connections are properly closed. This class includes methods for all common database queries needed by the recommender system. The query results are standardized, returning all data as python dictionaries.

2.3.2 Data Processing Pipeline

The ALSRecommender class serves as the data processing pipeline and consists of 3 main stages.

Initialization Stage

1. The recommender is initialized with passed through or default hyperparameters. These hyperparameters are discussed in more detail in Section 2.4.
2. A model may optionally be loaded/saved, which gives the ability to avoid having to rebuild the model each time.

Model Building Stage

If a model is loaded in the prior stage, this stage is skipped.

1. The system retrieves the deck data from the database
2. All unique card IDs are identified, filtering out basic lands
3. A sparse deck-card matrix is created where the rows represent decks, cards represent cards, and values represent the confidence levels.

4. The Alternating Least Squares algorithm performs matrix factorization to produce the card-factors and deck-factors

Recommendation Stage

- The user provides their commander and any cards they may already have in their deck.
- A deck vector is created using a weighted combination of the average deck vector of decks with the same commander (30%) and the average vector of existing cards in the deck provided by the user (70%)
    - This implementation uses a weighted average of known decks with the commander in an attempt to overcome the "cold-start" problem, where there is not enough context to make accurate predictions. Cards already in the deck are weighed higher to help provide more personalized recommendations as more cards are added to the user's deck.
- The dot product of this deck vector and the card-factors is taken to calculate a score for each card. This score is the likelihood that card belongs in the user's deck.
- Once these scores are returned, additional filtering is applied to:
    - Remove the cards already in the deck, including the commander
    - Only keep cards matching the commanders color identity
- The recommendations are then sorted by their score and returned to the user.

2.4 Hyperparameter Tuning

An ALSEvaluator class was created to evaluate different hyperparameter configurations for the final model. This class takes a list of different hyperparameters to test ranges of combinations of parameters. These hyperparameters included:

- **Latent Factors** - The number of hidden features for cards and decks
- **Regularization** - Prevents overfitting and increases generalization of the model
- **Alpha** - Confidence scaling of the model to increase the importance of implicit feedback (card inclusion).
- **Iterations** - The amount of times the ALS algorithm alternates between fixing card and deck matrices
- **Seed Percentage** - How much of the deck is used as seed cards

For each hyperparameter configuration, a model was built using a random set of 200,000 decks, which were controlled by a random seed to ensure the same decks were used each time. Each of these decks were tested against a set of test decks, which were split based on the seed percentage. These seed cards would imitate the cards provided by the user as input. The remaining cards, which are not being used as seed cards, would be “target cards” which would be compared to the recommended cards to evaluate the quality of the recommendations produced by the system.

Once the models were created, they were evaluated based on the following metrics:

- **Precision**: the proportion of recommended cards appearing in the target cards
- **Recall**: the proportion of target cards appearing in the recommended cards
- **F1 Score**: Harmonic mean of precision and recall, providing a balanced assessment of precision and recall

- **Mean Reciprocal Rank (MRR)**: How earlier the target cards appear in the recommendations, measuring the position of highly relevant cards.
- **Precision@k**: Precision for the top **k** recommendations, measuring the quality of the recommendations in the first **k** recommendations.

Multiple evaluation runs were performed using different hyperparameter configurations informed by the results of the previous rounds of testing. This approach allowed me to see how each parameter impacted the recommendation quality and determine an optimal hyperparameter configuration set. Tableau was used as a visualization tool to help interpret the results of these tests, which is further discussed in Chapter 3.

2.5 Conclusion

This chapter detailed the methodology used to create the Magic the Gathering Commander deck recommender system. It covered data collection and processing, system design, and the tuning of model hyperparameters. The next chapter will discuss the results of tuning the hyperparameter and the final model.

Chapter 3 - Analysis and Discussion

3.1 Introduction

This chapter will discuss how the final model was created, using the results of the hyperparameter tuning discussed at the end of chapter 2. It will show the final model was obtained, which was trained on 200,000 decks and tested against 20,000 decks. It will also show the impact of each hyperparameter on model performance by analysing the performance metrics F1 Score, Mean Reciprocal Rank (MRR), and Precision@5 (P@5).

Trying to visualize many parameter configurations simultaneously can lead to overly complex, and potentially uninterpretable, visualizations. Therefore, the visualizations in this chapter will have certain hyperparameters intentionally fixed. This approach provides a way to isolate the individual impact of specific parameters, allowing for a clearer interpretation of how each parameter affects the model's performance.

3.2 Hyper Parameter Tuning Results

3.2.1 Tuning Run 1

This initial run systematically tested each combination of the following hyperparameter values: regularization (0.5, 1, 2), alpha (10, 20, 30), and iterations (20, 25). This resulted in 18 distinct ALS models, which were all evaluated using F1 score, MRR, and P@5. The number of factors was set to 250 and a fixed seed percentage of 40%. This initial factor count was chosen as a starting point, as it seemed like a good enough factor count to capture meaningful relationships while also keeping the training time low. The goal was to establish a good starting point for my parameters. The results of this test can be seen in Table 1 below.

Factors	Reg	Alpha	Iter	F1 Score	MRR	P@5
250	0.5	10	20	0.3128	0.0571	0.5850
250	2	10	20	0.3256	0.0591	0.5898
250	1	10	20	0.3209	0.0589	0.6004
250	0.5	10	25	0.3136	0.0573	0.5866
250	2	10	25	0.3258	0.0591	0.5910
250	1	10	25	0.3223	0.0593	0.6038
250	0.5	20	20	0.3072	0.0561	0.5772
250	1	20	20	0.3137	0.0576	0.5892
250	2	20	20	0.3235	0.0592	0.5986
250	0.5	20	25	0.3081	0.0564	0.5768
250	1	20	25	0.3150	0.0578	0.5908
250	2	20	25	0.3244	0.0594	0.6004
250	0.5	30	20	0.3057	0.0557	0.5698
250	1	30	20	0.3102	0.0568	0.5820
250	2	30	20	0.3188	0.0585	0.5966
250	0.5	30	25	0.3063	0.0559	0.5710
250	1	30	25	0.3115	0.0571	0.5856
250	2	30	25	0.3197	0.0586	0.5958

Table 1 - Results from Tuning - Run 1

Figure 6 visualizes how regularization values (0.5, 1, 2) affect model performance, when keeping latent factors fixed at 250 and iterations fixed at 20. The data used in this chart can be seen in Table one, highlighted in yellow. This chart shows the F1 score consistently improving as regularization increases from 0.5 to 2 across all alpha values. This suggests the model may be prone to overfitting at lower regularization levels. MRR follows a similar positive trend, though less pronounced when moving from regularization 1 to 2, indicating there may be some diminishing returns. The effects on P@5 showed a different pattern, at an alpha value of 10, the peak performance was observed at a regularization level of 1. However, when alpha was 20 and

30, peak performance was observed to be at regularization level 2. This suggests that when the implicit feedback is weighed more heavily, stronger regularization is necessary to prevent overfitting. We can also see a general trend of performance increasing with lower alpha level. Based on this analysis, I chose to focus future tuning runs on higher regulation levels and lower

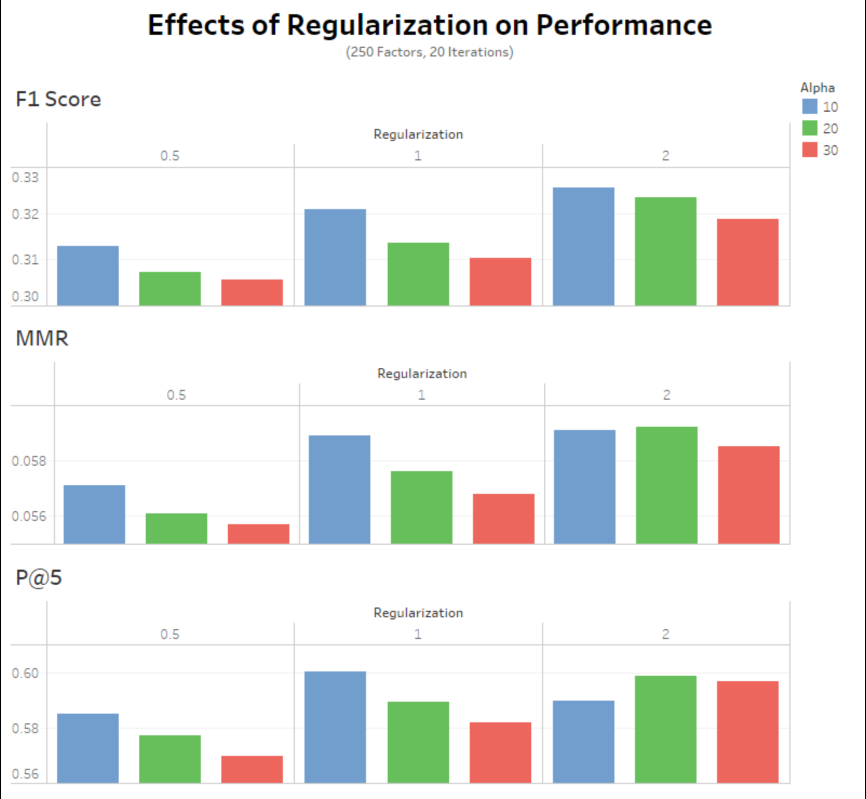


Fig. 6 - Effects of Regularization on Performance

alpha values.

3.2.2 Tuning Run 2

Based on the results from Run 1, I decided to focus on higher regularization values and further refine the alpha values. This run also investigated increasing the number of latent factors. This run tested each combination of the following hyperparameter values: factors (300, 350), regularization (1, 2), alpha (5,10,15), and iterations (20, 25). This resulted in the creation of 24 distinct models. The results of Run 2 can be seen in Table 2 below.

Factors	Reg	Alpha	Iter	F1 Score	MRR	P@5
300	1	5	20	0.3241	0.0598	0.6096
300	2	5	20	0.3167	0.0579	0.5820
300	1	10	20	0.3214	0.0594	0.6048
300	2	10	20	0.3283	0.0602	0.6044
300	1	15	20	0.3159	0.0585	0.5996
300	2	15	20	0.3263	0.0601	0.6118
300	1	5	25	0.3248	0.0599	0.6090
300	2	5	25	0.3174	0.0581	0.5826
300	1	10	25	0.3227	0.0597	0.6052
300	2	10	25	0.3287	0.0603	0.6076
300	1	15	25	0.3171	0.0588	0.6006
300	2	15	25	0.3276	0.0602	0.6098
350	1	5	20	0.3275	0.0605	0.6158
350	2	5	20	0.3211	0.0588	0.5946
350	1	10	20	0.3219	0.0595	0.6058
350	2	10	20	0.3305	0.0609	0.6194
350	1	15	20	0.3163	0.0587	0.6010
350	2	15	20	0.3270	0.0608	0.6144
350	1	5	25	0.3277	0.0605	0.6156
350	2	5	25	0.3208	0.0590	0.5970
350	1	10	25	0.3230	0.0598	0.6082
350	2	10	25	0.3304	0.0610	0.6194
350	1	15	25	0.3175	0.0589	0.6042
350	2	15	25	0.3281	0.0609	0.6150

Table 2 - Results from Tuning - Run 2

These results revealed several trends in the effect of the number of metrics. First, when looking at the result from increasing the number of factors, using regularization of 2, alpha of 10, and iteration count of 25 (highlighted in yellow), shows an increasing performance across all performance metrics. Comparing the results from test one, when using 250 factors and comparable regularization, alpha, and iteration count, the F1 score increased from 0.326 to 0.33

(1.2% increase), MRR increased from 0.059 to 0.061 (3.4% increase), and P@5 increased from 0.59 to 0.62 (4.7% increase). These trends can be seen in Figure 7.

The second insight was on the effects of alpha and regularization on performance which can be seen in Figure 8. We can see that F1 and MRR peak at regularization 2 and alpha 10. P@5 continues to increase slightly beyond 10, but with diminishing returns. Due to the downward trend in F1 score and MRR when increasing the alpha past 10, I opted to use an alpha value of 10 in future tests.

The last insight I gained was learning that there was only a slight increase in performance between 20 and 25 iterations. This can be seen in Figure 9. This led me to continue testing with only

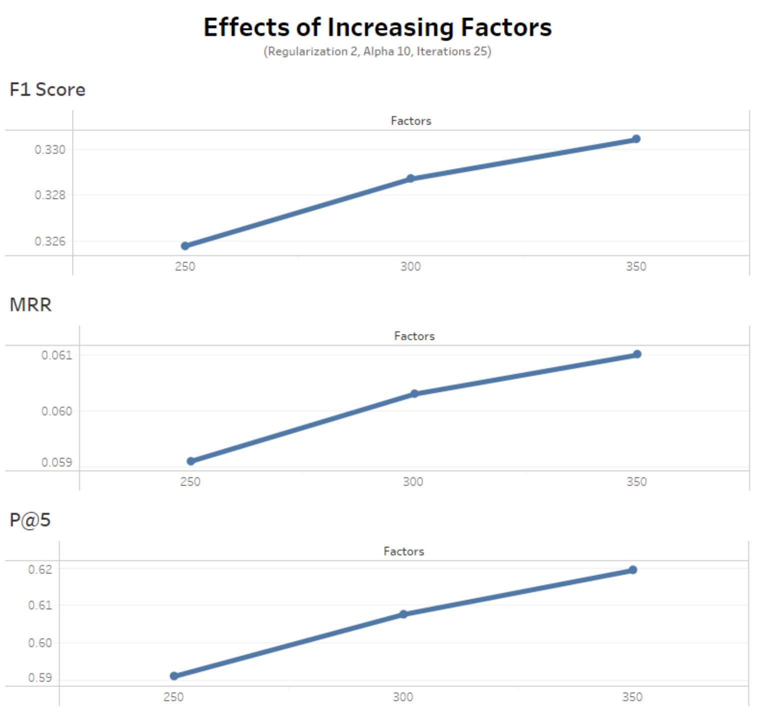


Fig. 7 - Effects of Increasing Factors on Performance Metrics

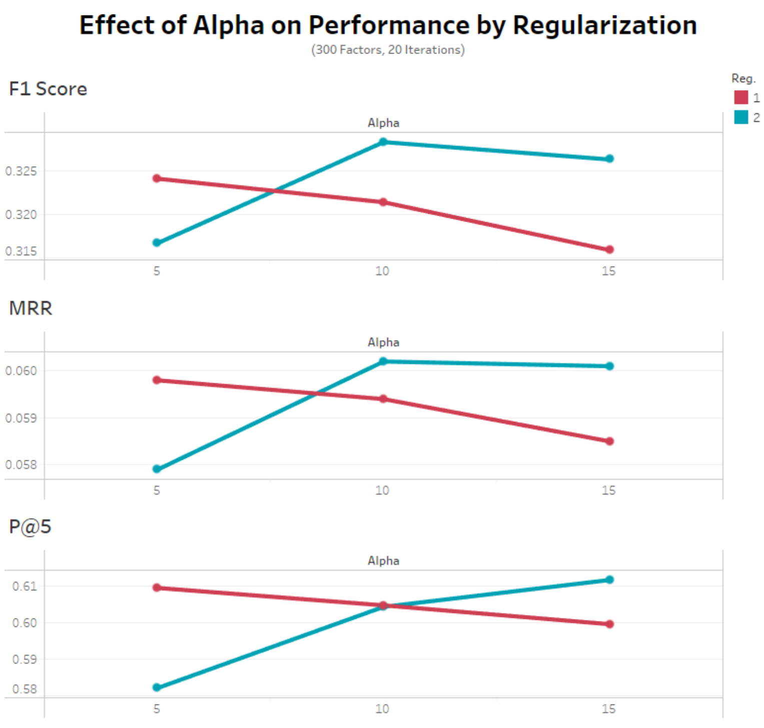


Fig 8. - Effects of Alpha on Performance Metrics by Regularization

25 iterations. Even though the increase was marginal, the computational cost was minimal.

Based on these results, I chose to continue my tuning with a regularization of 2, alpha of 10, and iteration count of 25.

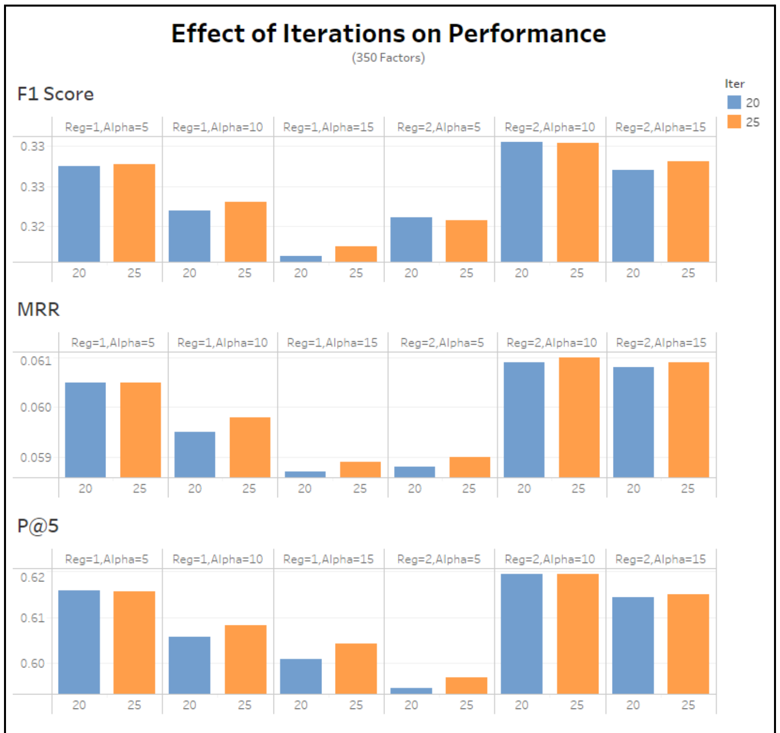


Fig 9. - Effects of Iterations on Performance Metrics

3.2.3 Tuning Run 3

For this run, I decided to increase my factor count beyond 350, testing on 400-600 factors. The result of this run can be seen in Table 3.

Factors	Reg	Alpha	Iter	F1 Score	MRR	P@5
400	2	10	25	0.3322	0.0615	0.6242
450	2	10	25	0.3322	0.0619	0.6290
500	2	10	25	0.3313	0.0625	0.6312
550	2	10	25	0.3325	0.0625	0.6376
600	2	10	25	0.3312	0.0628	0.6390

Table 3 - Results from Tuning - Run 3

Figure 10 combines the results from multiple runs to visualize the effects of increasing the factor count from 250 to 600. MRR and P@5 show continuous improvement, which begins to level off around 550 to 600 factors. This suggests that increasing the number of factors gives the model a better ability to rank the most relevant cards higher.

F1 score appears to level off around 400 factors, indicating that the overall quality of recommendations begins to plateau around this point. This shows how different aspects of the recommended cards change based on model complexity.

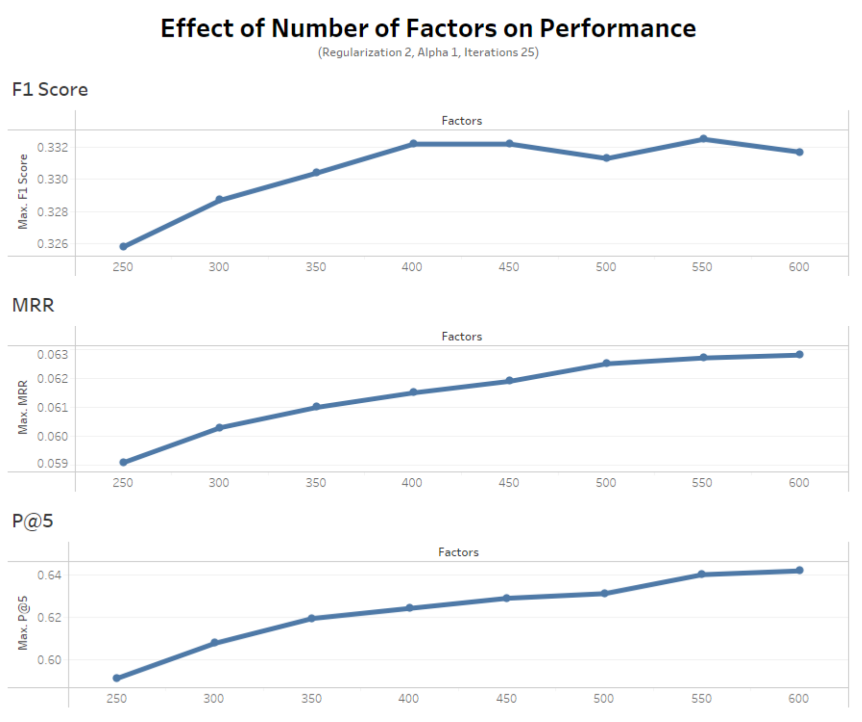


Fig. 10 - Effect of Increasing Factors from 250 to 600 on Performance Metrics

Based on these results, I chose to continue testing using 600 factors, as at this point there seemed to be a good balance between all of the performance metrics.

3.2.4 Tuning Run 4

For this run, I wanted to see if further regularization increased the performance of the model. For this run I continued with a factor count of 600, Alpha of 10, and Iter count of 25. The regularization values tested were 2, 2.25, 2.5, and 3. The results from this run can be seen in Table 4.

Factors	Reg	Alpha	Iter	F1 Score	MRR	P@5	P@10
600	2	10	25	0.3330	0.0628	0.6381	0.5758
600	2.25	10	25	0.3326	0.0629	0.6428	0.5785
600	2.5	10	25	0.3329	0.0631	0.6426	0.5799
600	3	10	25	0.3326	0.0629	0.6418	0.5797

Table 4 - Results from Tuning - Run 4

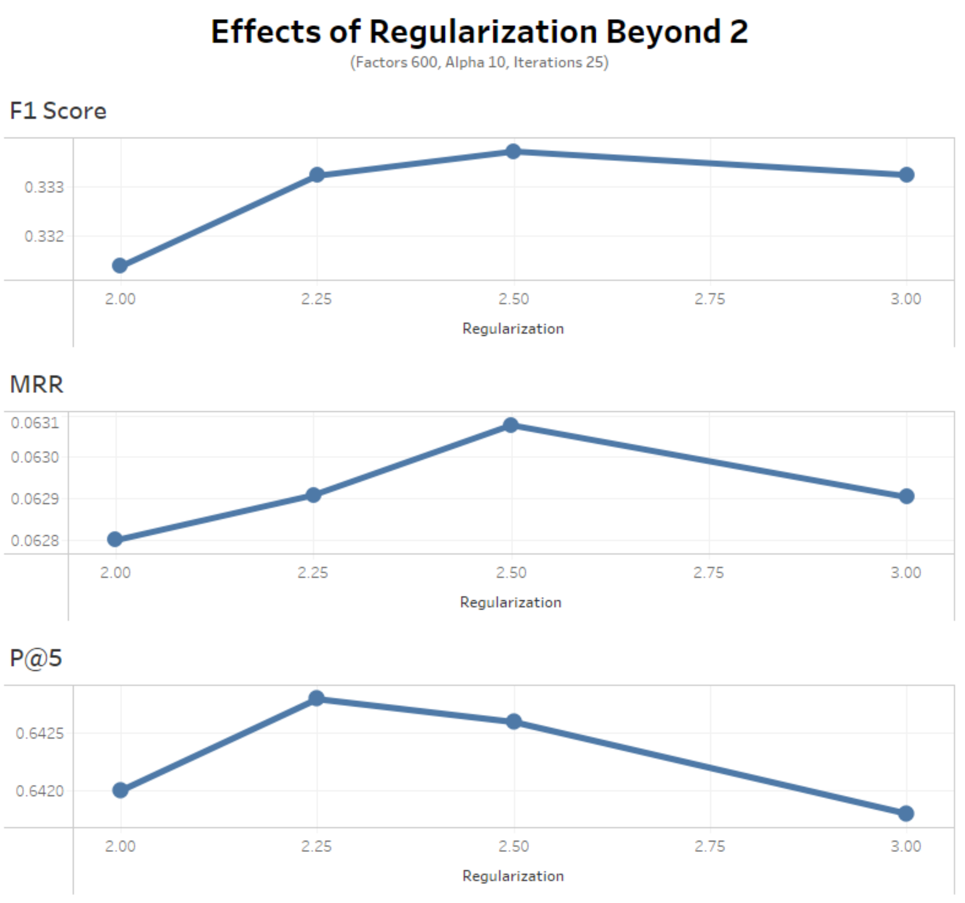


Fig. 11 - Effects of Regularization from past 2

The effects of regularization from this run can be seen in Figure 11. From this run, I could see there was an increase in F1 score and MRR up until regularization 2.5. P@5 peaked at regularization 2.25, and P@10 peaked at 2.5. This suggests that using a regularization of 2.5 provides better overall recommendations when compared to using a regularization of 2.25. However, when using regularization of 2.25, the early recommendations are more relevant.

3.2.5 Tuning Run 5

For my final tuning run, I decided to do further testing on the difference between regularization 2.25 and 2.5. For this test, I chose to use different seed percentages of 0%, 25%, and 50% in order to see how this regularization impacts recommendation quality at varying stages of the deck building process. The results from this can be seen in Table 5.

Factors	Reg	Alpha	Iter	Seed	F1_Score	MRR	Precision@5	Precision@10
600	2.25	10	25	0%	0.2483	0.0262	0.3913	0.3538
600	2.25	10	25	25%	0.3662	0.0546	0.6721	0.6173
600	2.25	10	25	50%	0.2994	0.0703	0.6143	0.5393
600	2.5	10	25	0%	0.2375	0.0255	0.3822	0.3447
600	2.5	10	25	25%	0.3654	0.0543	0.6670	0.6143
600	2.5	10	25	50%	0.3000	0.0705	0.6152	0.5411

Table 5 - Results from Tuning - Run 5

From these results, we can see the effects of the cold start problem. The performance metrics all increase dramatically from 0% seed percentage to 25% seed percentage. Comparing regularization 2.25 to 2.5, we can see that 2.25 performs better during a cold start. The F1 score increases by 4.5%, MRR increases 2.7%, P@5 increases by 2.4%, and P@10 increases by 2.7%.

At 25% seed regularization of 2.25 slightly out performs regularization 2.5, however the increase is all less than 1% for all metrics. The largest difference is in P@5, where regularization 2.25 out performs regularization 2.5, increasing by 0.76% from 0.667 to 0.672. There is very little difference between regularization 2.25 and 2.5 at 50% seed.

3.3 Conclusion

Based on the results of this tuning, I decided on a final model built on 200,000 decks with the configuration set of 600 latent factors, Regularization of 2.25, Alpha of 10, and 25 Iterations. When tested against 20,000 decks 40% seed, this configuration resulted in an F1 Score of 0.333, MRR of 0.063, P@5 of 0.643, and P@10 of 0.579.

Chapter 4 - Conclusions

4.1 Summary

My goal was to set out to create a tool to assist both new and experienced Magic the Gathering players in constructing a deck for the Commander format. This task involves the player choosing 99 cards from a card pool of over 28,000 unique cards that adhere to their chosen commander's color identity. This tool used collaborative filtering through Alternating Least Squares matrix factorization to identify the relationship between cards in Commander decks.

The final model was created using data from 200,000 decks using 600 features, regularization of 2.25, alpha of 10, and 25 iterations. These model parameters were chosen by systematically evaluating different hyperparameter sets to find an optimal configuration. With this configuration, when tested against 20,000 decks at a 40% seed percentage, it was able to achieve an F1 score of 0.333, MRR of 0.063, P@5 of 0.643, and P@10 of 0.579. The F1 score of 0.333 indicates the model has a good balance between the precision and recall. The P@5 indicates that roughly two-thirds of the top 5 cards being recommended appeared in the target set of cards. Additionally, the P@10 indicates that over half of the top 10 recommended cards appeared in the target set of cards.

4.2 Future Work

I fully intend to continue developing this project and have many future goals:

1. I would like to continue the web scraping process, giving me access to more deck lists. This would require a process for adding new cards to the database as they are released.
2. Improve the evaluation process, as a new model would need to be trained as new cards get released. The evaluation process as it stands is currently a single threaded application. Splitting the evaluation step between multiple workers will decrease the time it takes during the tuning stage dramatically.
3. Once the evaluation process is optimized, I would continue testing different hyperparameters at a larger number of seed percentages. This would potentially lead to an implementation where there are multiple models tuned at different seed percentages. Based on the stage at which the user is in their deck building process, the system would determine which model to use.
4. Implement an optional filter to only recommend certain card type(s). For example, if a user really only wanted to be recommended Creature cards, they would have that option.
5. Use card price data to implement an optional restriction on card price or total deck price.
6. Implement a frontend for the recommender system. This will provide the users with a web-based interface for interacting with the recommender system. I have included a basic UML diagram (Figure 12) of how I envision the fully implemented recommender system would operate in a production environment.

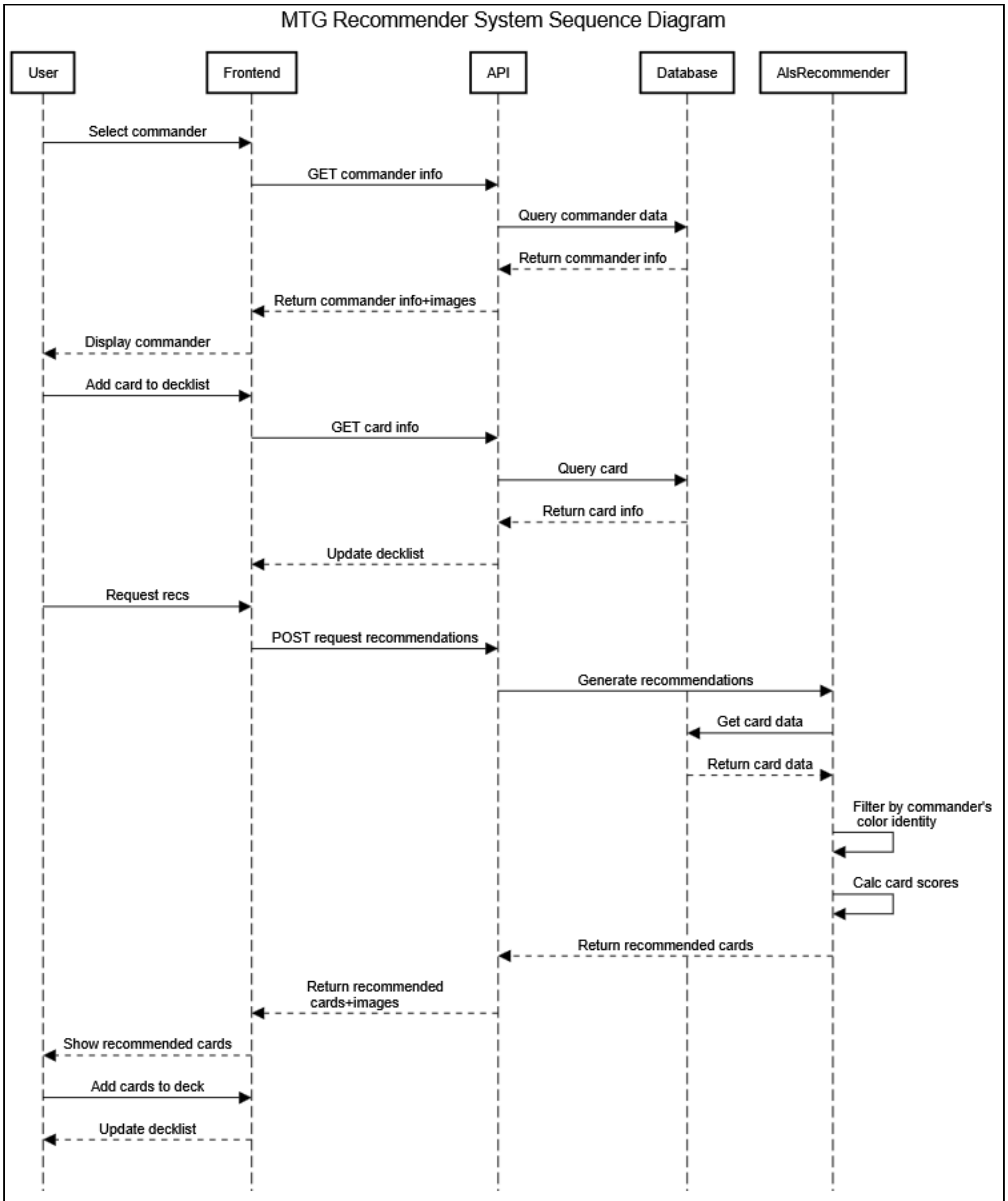


Fig. 12 - MTG Recommender UML Diagram

References

1. Burke, R., Felfernig, A., Goker, M. (2011). Recommendation Systems: An Overview. *AI Magazine*
2. Sohail, S., Siddiqui, S., Ali, R. (2017). Classification of Recommender Systems: A review. *Journal of Engineering Science and Technology Review*
3. Patel, B., Palak D., Panchal, U. (2017). Methods of Recommender System: A Review. *ICIIECS*
4. Victor. (2017) ALS Implicit Collaborative Filtering. *Medium.* https://medium.com/radon-dev/als-implicit-collaborative-filtering-5ed653ba39fe (Retrieved March, 2025)
5. Hu, Y., Koren, Y., Volinsky, C. (2009). Collaborative Filtering for Implicit Feedback Datasets
6. Moxfield. (2024) https://moxfield.com/ - Datasource for completed decklists
7. Scryfall. (2024) https://scryfall.com/ - Datasource for card information and images

Appendix

Appendix A:

The code for mtg_database.py, als_recommender.py, als_evaluator.py can be found on GitHub using the following link: https://github.com/bdeni-ramapo/mtg_recommender

Appendix B:

I created a mockup of how I envisioned the recommender system working in a production environment, which can be seen in Figure A1. It uses React for the frontend, Tailwind CSS for styling, and FastAPI for the backend. On the top left, there is a section for the user to input their chosen commander. Below that, there is an optional section where the user

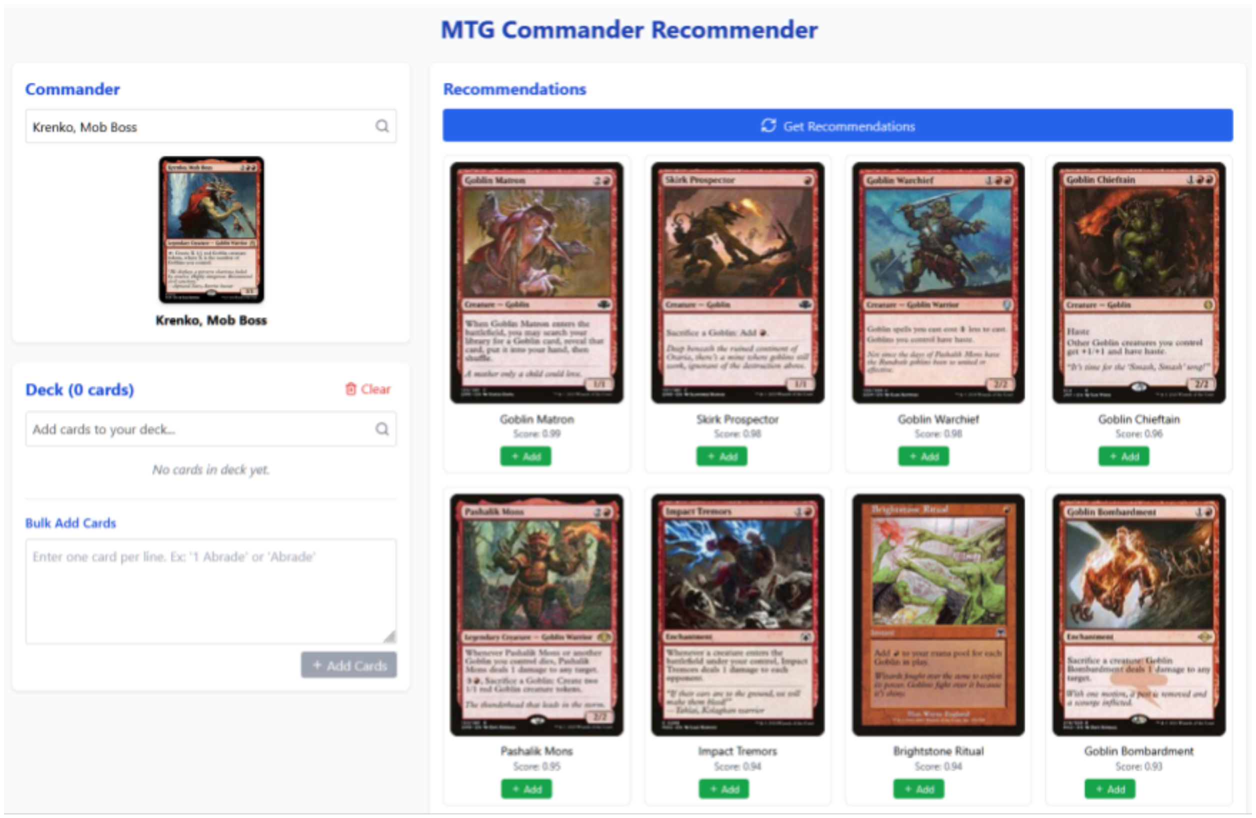


Figure A1 - Mockup of Recommender System

may input any cards they may already have in their deck. On the right side, there is a "Get Recommendations" button. Once clicked, the recommendations and their scores are generated and output to the user in the section below. Additionally, there is an 'Add' button below each recommendation which the user can click which will add that card to their decklist. Once one or many cards are added, the user may then click the 'Get Recommendations' button again to generate new recommendations based on the new decklist.

The example in Figure A1 shows that the user has chosen the card 'Krenko, Mob Boss' as their commander, a Red commander which can be seen closer in Figure A2. From the rules text, we can see Krenko has an effect that creates 'Goblin' tokens equal to the number of 'Goblin' creatures you control. Based on that clear synergy, we would expect the recommendations to include a lot of Goblin creatures.

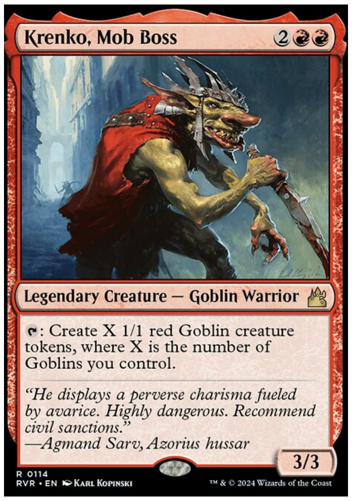
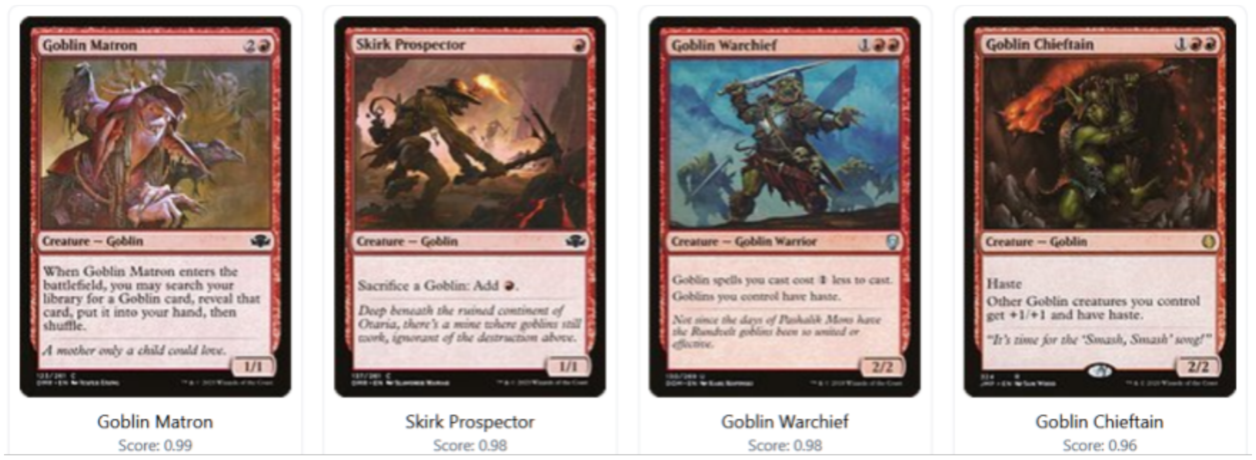

Krenko, Mob Boss
2
Legendary Creature — Goblin Warrior
: Create X 1/1 red Goblin creature tokens, where X is the number of Goblins you control.
"He displays a perverse charisma fueled by avarice. Highly dangerous. Recommend civil sanctions."
—Agmand Sarv, Azorius hussar
3/3
R 0114
RVR • EN Karl Kopinski
™ & © 2024 Wizards of the Coast

Figure A2 - Krenko, Mob Boss

Figure A3 shows an enlarged version of the first row of cards from Figure A1. We can see there are many Goblin cards being recommended, indicating we are getting useful recommendations for this commander. Additionally, we can see all of the recommended cards are Red, indicating that the color identity restriction put in place is being adhered to correctly.



Goblin Matron
Score: 0.99

Skirk Prospector
Score: 0.98

Goblin Warchief
Score: 0.98

Goblin Chieftain
Score: 0.96

Figure A3 - Recommendations for Krenko