DEVELOPMENT AND EVALUATION OF THE GRANDMASTER OPENINGS MOBILE APP: A COMPREHENSIVE SOLUTION FOR ENHANCING CHESS OPENING KNOWLEDGE USING AI AND FLUTTER

Anjun Chen[1], Garret Washburn[2]

[1]Sage Hill School, 20402 Newport Coast Dr., Newport Beach, CA 92657
[2]Computer Science Department, California State Polytechnic University, Pomona, CA 91768

ABSTRACT

This research is intended to display and encapsulate a methodology for solving the problem of the desperate lack of information and resources for learning about and practicing opening moves in Chess. The methodology, GrandMaster Openings, comes in the shape of a mobile application available on both the Apple and Google Play stores and can be downloaded right now [1]. The GrandMaster Openings app seeks to give the user insightful information on opening moves in Chess, and even comes equipped with an AI chat feature with an AI GrandMaster trained on the same data that is on display for users to see within the app [2]. The most prominent technologies that were utilized to develop this application are ChatGPT's trainable chat model features and the Flutter mobile application development framework, as well as a few extensive chess match datasets. During development, the creation and implementation of a back-end server was necessary, as the transmission of quite extensive data and the housing of a big AI model became cumbersome to keep on a user's device, and proved to be quite challenging due to the new implications of a back-end server [3]. Within this essay are experiments that were conducted specifically targeted at the back-end server in order to discover any current or potential issues with it. Ultimately, after proper development and thorough review, the GrandMaster Openings mobile application is a great resource for those looking to learn more about Chess openings, get immediate feed from a trained AI model GrandMaster, and review excellently collected real life game data.

KEYWORDS

Chess, Opening, AI, LLM

1. INTRODUCTION

In chess, the first move is, often unknowingly by the players, the most important move of the game, as it sets the pace of the game and locks in the players with their strategies. However, despite the importance of the opening move in a chess match, there are not many widely available resources for practicing or studying opening moves. This creates a big problem for the chess community, as those who seek to study opening moves don't always have the resources to do so despite the consequences - "Your opening can give away your style to your opponent and vice versa, thus, it gives you ample opportunity to play according to your opponent's style."

David C. Wyld et al. (Eds): CCSIT, NLPCL, AISC, ITE, NCWMC, DaKM, BIGML, SIPP, SOEN, PDCTA – 2024
pp. 123-132, 2024. - CS & IT - CSCP 2024 DOI: 10.5121/csit.2024.141712

Additionally, there are not many statistics on the efficacy of certain opening moves, and how they can impact the game or its players [4]. A common way to handle these types of statistical or prediction problems is to employ the use of AI models for handling big amounts of game data and making predictions. In a study already conducted that employed the use of a classification model, it "provided the highest accuracy of 66.91% when two attributes namely opening sequence and winning score are considered in the 20-moves dataset [5]. "Using AI could be beneficial for chess players to study opening moves, as the lack of resources for opening moves and the unemployed use of AI models for this task is wasted potential. Utilizing AI train on real life game data, specifically trained on opening move data, can give new perspectives on opening moves and provide useful insights for players.

The three other methodologies investigated in this paper are Chess.com, ChessPathways.com, and the Chessable mobile application [6]. All three of the reviewed methodologies are different products that aim to be services of teaching not only opening moves in chess, but also general gameplay and strategy. While each one does aim to provide insight to chess gameplay, we did find some issues during the review that we seeked to improve upon in our mobile application. For all of these methodologies, the biggest issue we found was the lack of support and significant lack of any reviewed gameplay data. The GrandMaster Opening's application invests a lot of resources in maintaining and providing quality game review data for the user to reflect upon and use during training [7]. Another big lacking point that these methodologies tend to not include is any real time and quick support. GrandMaster Opening's comes equipped with a custom AI model designed to provide quick and insightful chess advice or information tailored for the individual user [8].

Our solution to this problem is to employ the use of an AI model that is trained on real life game data, that is designed to give feedback on opening moves based on statistical analysis. The frontend of this AI model will be a downloadable mobile application that users can download from respective app stores. This accessibility to the application and the AI model will solve the issue of resources not being readily available to players, as it will give them direct access to existing game data and an AI model specifically designed to interpret that data. Additionally, the application will be entirely free, as opposed to other resources that charge a fee or monthly subscription to have access to the same or lesser features. Because of the importance of openings, free resources will bring more accessibility to chess players, enabling a more informed player base. Additionally, the application is also intended to give practical game advice based on the statistics gathered from the game data it was trained on. This advice and analysis will provide users with a really great resource for practicing playing, as it is able to give advice in real time. The dynamic nature of the application allows for a more convenient and practical method of study, as opposed to other resources that are static and are not flexible depending on the situation. Overall, the application will be a great resource for those looking study opening moves, as well as those who are looking for immediate and direct feedback from the AI model.

After the initial development of the GrandMaster Openings mobile app, the research team conducted two thorough experiments in an attempt to root any potential issues or vulnerabilities in the application. The two experiments conducted within this paper were designed to test the response time of the back end server as well as the secure design of the AI model prompt input. In the first experiment, data was collected of how fast it took the backend server to receive the input from the user, create the prompt, get the response from the AI model, and send the data back to the user. We found that the average response time from the server was around 6.45 seconds, which we were satisfied with. The second experiment designed to test the security of the AI model we were also satisfied with. This experiment included the submission of prompts designed to try to get the AI model to return any response that is besides the instructions we gave to the model. We found that it was very difficult to have the AI model break character, as we

were unable to actually get any response from the model where it broke character or returned incorrect or useless data back.

2. CHALLENGES

In order to build the project, a few challenges have been identified as follows.

2.1. Effectively Using the Collected Game Data

One major component of the application proposed in this research paper are the backend functions designed to provide fuse AI and data to provide feedback on opening moves in chess. The most significant difficulty with creating this component of the project was effectively using the collected game data for predictions while also navigating an immense amount of data. Due to the nature of how big the game data was, effectively using it for predictions yielded quite a challenge, as efficiently using just enough data for a prediction was essential as to avoid unnecessary wait times in the backend. Additionally, deciding how much data to give to ChatGPT for analysis was also a challenge, as ChatGPT can have variable answers and perfecting the ratio of time efficiency and accuracy with the ChatGPT functionality was essential.

2.2. Embedding the Backend Functions Into the Server

Another major part of the application is the server which hosts the backend functions as a callable API. One of the most prominent challenges encountered with this component was actually embedding the backend functions into the server. As what the backend functions returns depends on what the function does, formatting the data in a manner that is easy to send across the internet is also very important, as the data returned from some functions is not conducive to effective transportation. Specifically, putting the data returned from the functions into a json format was essential, as the json format is not only very standard for data transportation, but also is convenient for maintaining structure and easy access on the user's application side.

2.3. Working With the Flutter Framework

Additionally, another aspect of the overall application that we encountered some challenges with is the user side Flutter application. Working with the flutter framework was challenging as it is moreso a framework as opposed to a language, and comes with its own unique attributes and challenges.Specifcally, maintaining the framework was quite difficult as there were often packages and updates that had to be installed which had their own dependencies associated with them. Creating a function to fetch data from our API server was also in it of itself challenging, as previously stated the backend functions could return very different outputs. The challenge came in in not only establishing a way to call each individual backend function, but also how to put all of the varying responses into one format that can be used anywhere in the application. The solution created was to simply store all responses as a string manipulate them in their separate areas after.

3. SOLUTION

The method outlined in this research paper, ChessGPT, is an application designed to provide data and AI backed analysis on opening moves in chess in real time. The three major components that make up the structure of ChessGPT are the backend functions, the api server, and the user side Flutter application. Upon opening the Flutter application, users are greeted by a splash screen that moves into the home screen. Inside the home screen, users are shown a Trending page and at the

bottom, a ‘Winning” tab which displays the topmost winning opening moves. From the homescreen, the user can either start a new AI chat or open the “New” tab and see even more of the newest data analytics on opening move data. For each opening move the user can inspect each by clicking on them and arriving at a “Details” screen for that specific opening move. Alternatively, the user can open an AI chat page from the homescreen, in which the user can create custom commands to the AI server to get data or ask for an AI predictive analysis on an opening move or even for advice on their current chess game [9]. Every time a user creates a request to the AI chatbot, the request is sent to the AI server hosted on render.com. Inside the server, depending on what functionality the user is requesting based on the command they inputted, the corresponding function is called which will either grab data from the database or ask ChatGPT a question based on the question from the user and data grabbed from the database. Upon formulating the response in a json format, the response is then sent back to the user who made the request and displayed onto their screen for their viewing.

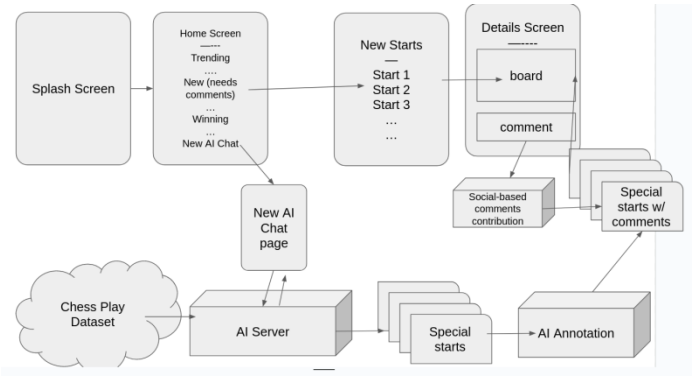
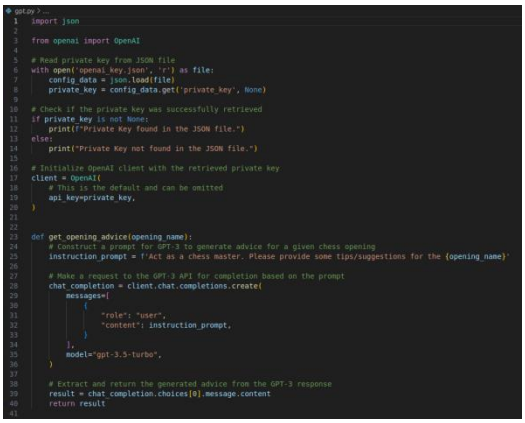


Figure 1. Overview of the solution

The backend functions for the Chess Opening Analyzer app serve the purpose of either grabbing data from the chess game datasets or making calls to ChatGPT in order to give the user helpful data and advice on improving their opening moves [10]. These functions rely on the ChatGPT api as well dataset manipulation concepts utilizing Python’s pandas library.

```python
import pandas as pd

# Set pandas display options
pd.set_option('display.max_columns', None)
pd.set_option('display.max_colwidth', None)

# Load the dataset from Kaggle
# Dataset download link: https://www.kaggle.com/datasets/datasnaek/chess/data
df = pd.read_csv('games.csv')
df.dropna()


# Get unique opening names
def get_opening_name():
    result = df['opening_name'].unique()
    return result.tolist()


# Get top N opening names by frequency
def get_top_n_opening_name(n):
    # Group the DataFrame by 'opening_name' and get the size (frequency) of each group
    result = df.groupby('opening_name').size()
    # Sort the result by frequency in descending order
    result = result.sort_values(ascending=False).reset_index(name='count')
    return result[:n]


# Get victory status statistics
def get_victory_status():
    # Group the DataFrame by 'opening_name' and 'victory_status' and get the size (count) of each group
    result = df.groupby(['opening_name', 'victory_status']).size()
    # Sort the result by count in descending order
    result = result.sort_values(ascending=False).reset_index(name='count')
    return result


# Get victory status statistics for a specific opening name
def get_victory_status_by_opening_name(opening_name):
    # Filter the DataFrame for rows with a specific 'opening_name'
    # Group the filtered DataFrame by 'opening_name' and 'victory_status' and get the size (count) of each group
    result = df.loc[df['opening_name'] == opening_name].groupby(
        ['opening_name', 'victory_status']).size()
    # Sort the result by count in descending order
    result = result.sort_values(ascending=False).reset_index(name='count')
    return result
```



```python
import json

from openai import OpenAI

# Read private key from JSON file
with open('openai_key.json', 'r') as file:
    config_data = json.load(file)
    private_key = config_data.get('private_key', None)

# Check if the private key was successfully retrieved
if private_key is not None:
    print(f"Private Key found in the JSON file.")
else:
    print("Private Key not found in the JSON file.")

# Initialize OpenAI client with the retrieved private key
client = OpenAI(
    # This is the default and can be omitted
    api_key=private_key,
)


def get_opening_advice(opening_name):
    # Construct a prompt for GPT-3 to generate advice for a given chess opening
    instruction_prompt = f'Act as a chess master. Please provide some tips/suggestions for the {opening_name}'

    # Make a request to the GPT-3 API for completion based on the prompt
    chat_completion = client.chat.completions.create(
        messages=[
            {

                "role": "user",
                "content": instruction_prompt,
            }
        ],
        model="gpt-3.5-turbo",
    )

    # Extract and return the generated advice from the GPT-3 response
    result = chat_completion.choices[0].message.content
    return result
```

Figure 2. Screenshot of code 1

The first major component we would like to address is the back-end server, and the backend functions within them. The backend functions, as part of the server, serve the purpose of taking the user's http request received by the server and performing their respective functionalities in order that the desired data may be returned back to the user's device. The list of backend functions includes get_opening_advice, get_opening_name, get_top_n_opening_name, get_victory_status, and get_victory_status_by_opening_name. All of the functions, minus get_opening_advice, use pandas to grab data from the chess opening dataset with the intention of providing the user with information relevant to the opening name they are querying for. However, the get_opening_advice method is intended for the AI chat feature within the app. The get_opening_advice function uses the OpenAI prompt functionality from the OpenAI library to send a request to ChatGPT from the Flask server with the question that the user sends in their request. The get_opening_advice function first sets the persona of ChatGPT to a chess GrandMaster, so that it's responses are in theme and helpful.

Within the overview of the GrandMasterOpenings mobile application, the backend server serves a critical as being the data and content center. All of the chess dataset data and the custom AI model prompts are kept on the back-end server, and for any of the data to be used the user must perform a request to the server.

```python
# Route to get trending data
@app.route('/get_trending_data')
def get_trending_data():
    return jsonify(trending_data)

# Route to get new posts data
@app.route('/get_new_posts')
def get_new_posts():
    return jsonify(new_posts)


# Route to fetch a list of all opening names.
@app.route('/opening_name_list')
def fetch_opening_name():
    opening_name_list = get_opening_name()
    return jsonify(opening_name_list)


# Route to fetch the top N opening names by frequency.
@app.route('/top_n_opening/<number>')
def top_opening(number):
    top_n_opening = get_top_n_opening_name(int(number))
    result = top_n_opening.set_index("opening_name").T.to_dict('dict')
    return jsonify(result)


# Route to fetch victory status statistics for all opening names.
@app.route('/victory_status/')
def victory_status():
    victory = get_victory_status()
    result = victory.set_index("opening_name").T.to_dict('dict')
    return jsonify(result)


# Route to fetch victory status statistics for a specific opening name.
@app.route('/victory_status_by_opening_name/<opening_name>')
def fetch_victory_status_by_opening_name(opening_name):
    victory_status_by_opening_name = get_victory_status_by_opening_name(opening_name)
    data = victory_status_by_opening_name.values.tolist()
    result = {'statistics': data}
    suggestions = get_opening_advice(opening_name)
    result['suggestions'] = suggestions
    return jsonify(result)
```

Figure 3. Screenshot of code 2

The back-end server is ran in Python and uses the Flask library to create a flask server. Using the Flask library, the back-end server creates a variety of routes for the user to grab specific content or engage in different backend utilities such as the AI model prompts or even querying the dataset. As seen in the limited screenshot above, there are some routes defined to return quite a variety of data to the user. The first '/get_trending_data' is intended for the user to make a request towards to receive the most up to date trending data that we are including on the trending page. We update this data regularly to keep the user up to date with Chess news. One of the route below in the screenshot, the '/get_opening_name_list' route, serves the purpose of using the get_opening_name backend function and returning the list back to the user. The rest of the routes underneath in the screenshot all serve a similar purpose of using the backend functions to grab data and return said data back to the user.

On the opposite end of the backend server are the user devices and the GrandMasterOpenings mobile application. The mobile application, written in Dart using the Flutter framework, serves the purpose of neatly receiving, formatting, and displaying the data and responses received from the server in a manner that is easy to use and understand for the user.

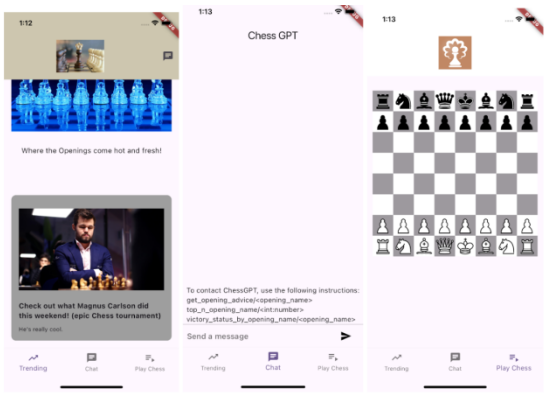


Figure 4. Screenshot of the APP

```dart
//Function for fetching data from our api
Future<void> fetchData(String path) async {
  //declare url
  var url = Uri.parse('https://chess-opening-analyzer.onrender.com/$path');

  try {
    // http GET request
    var response = await http.get(url);

    //check if response was successful
    if (response.statusCode == 200) {
      ChatMessage message = ChatMessage(message: response.body, isUserMessage: false);

      setState(() {
        _responseData = response.body;
        _message.insert(0, message);
      });
    } else {
      print("Request failed with status: ${response.statusCode}");
    }
  } catch (e) {
    print("error: $e");
  }
}
```

Figure 5. Screenshot of code 3

The GrandMasterOpenings application front-end was built using the Flutter framework, however, the most important part of the application has to do with how the data is sent and received from the back-end server. As such, depicted above is the code necessary for making the http requests to the back-end server, the fetchData function. The fetchData receives the path parameter as a string and will make the request to the back-end at that path. This makes it easier to have one

function that can grab data, values, or just about anything and return the data back to the original place the function was called. In the respective places where the fetchData function is called, such as the trending page or the chat feature, the data is returned back to that place and whatever parsing needs to be done can be carried out. Once that data has been returned, the data can then be displayed on the screen using the Flutter framework widgets.

4. EXPERIMENT

4.1. Experiment 1

As the GrandMaster Openings mobile application relies so heavily on a back-end server, during the app development process we became concerned about the stability and speed in which the back-end server can return data. As such, we decided to conduct our experiment on the speed of the back-end server.

To test the reliability and effectiveness of the back-end server, we set up a simple experiment designed to test the response time of the back-end server. We set up a simple excel sheet table, on the left the number of attempts and on the right the time in seconds of which it took for the back-end server to return a response. To properly conduct this experiment, we had one member using the mobile app and sending chat requests to the backend server and another member using a stopwatch to start and stop the timer when the response was sent and then received.

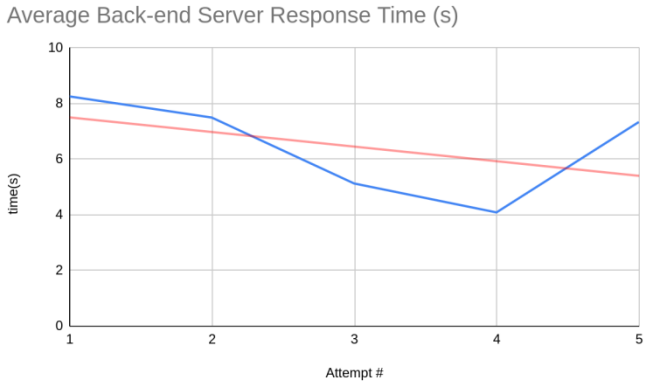


Figure 6. Figure of experiment 1

After conducting the experiment, we found that the average response time from the back-end server was around 6.454 seconds. The quickest response time was attempt #4 at 4.08 seconds and the longest response time was attempt #1 at 8.25 seconds. After collecting the complete set of data, the first surprise we came upon was how attempt #5 took a longer time to respond than attempt #4. We felt this way because originally we thought that the server was ‘warming up,’ or that is at least getting quicker because it was now awake and being used. However, after some thought and review, we found that the AI model generation that was returned in attempt #4 was half the length of that of attempt #5, incentivizing us to believe that the content being generated or returned affects the speed of the response time. Granted, this experiment was performed in an environment with a quality internet connection; ultimately a user’s response time from the server will vary upon their internet connection.

4.2. Experiment 2

Another issue we are slightly concerned of is the potential for a prompt injection[12] to be performed against the custom AI model. As the input given to the AI model is directly created by the user, this naturally opens the AI up to it, and as such proper measures need to be taken.

To test whether the AI Smart Route mobile app is susceptible to prompt injection, we have designed an experiment and a few prompts to submit. The goal of the experiment is to use a few strategies, such as asking the AI model to disregard previous instructions or to take on a new persona, and try to get the AI model to respond in a way or do something separate from what is intended. In our tests, we try to get the AI model to write a paragraph about a separate topic from chess, as that directly violates its rules we established when we created the AI prompt.

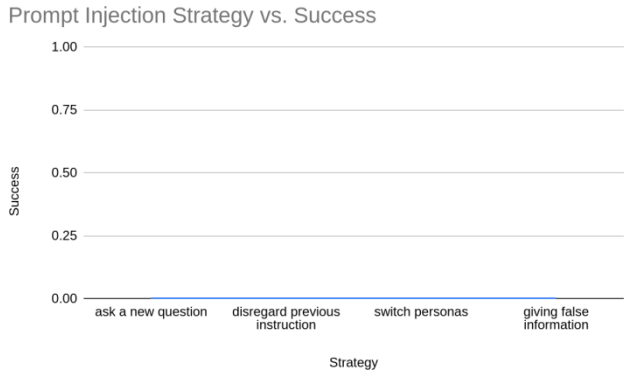


Figure 7. Figure of experiment 2

After conducting the experiment, we were quite surprised and happy with the results we collected as we found none of the major prompt injection strategies we tested yielded positive results. The most surprised we were definitely had to be when the switch personas strategy didn't work, as that seemed to be the most compelling and had us believe in it's effectiveness. We were also quite surprised that the giving of false information strategies weren't effective, as not only did it point out that as a GrandMaster it couldn't abide by the false information, it also corrected the false information we gave it about chess. Overall, we were satisfied with the results and believe that the reason none of them were effective was due to how specific the prompt instructions were that we created during the server development process. Overall, if we had to change anything, we would likely improve the rules declared in the prompt to perhaps cover a few more attack vectors.

5. RELATED WORK

Probably the most prominent and well known Chess resource, not only for openings but for anything chess in general, is chess.com [11]. Chess.com is a website that user's can visit and learn, talk about, and play Chess. Chess.com offers a lot of tutorials for opening moves, however, it's biggest noticeable issue is the fact that no win-rate data, or any data in general, is displayed for the opening moves. This is a big drawback for chess.com, however, the GrandMaster Openings mobile app offers an extensive amount of data from a set of proven datasets to support the win rate of opening moves used in competitive chess play.

Another methodology that seeks to solve the problem in the shape of a mobile app is Chessable [12]. Chessable is a web service that also has a mobile that seeks to provide courses and knowledge for the user to learn how to play a better game of chess, this includes lessons on

opening moves. However, the biggest thing we noticed during our research was that Chessable proved to be quite expensive, having a monthly subscription fee of $11.99, as well as extra courses from GrandMasters that ranged from $20 to $200 . Finally, we did also take note Chessable does not provide any match data. GrandMaster Openings does a better job of providing real game data as well as tailored responses from the AI model for the user to help their game, not just general advice from a GrandMaster.

Chesspathways.com is another methodology that seeks to act as a chess community website that allows individuals to meet and discuss chess, as well as put into contact users with the GrandMasters [13]. Overall the website looks great, it allows users to come learn from GrandMaster's, directly seek their support, and communicate with the chess community. However, while the site does include some teaching resources and GrandMaster's, an individual that is a part of the community must wait to contact the GrandMaster and either email them or set up a meeting. Meaning, the individual does not have direct and immediate communication with the GrandMaster they are seeking to learn from. GrandMaster Openings seeks to remedy this by having our AI model be able to communicate with the user at any point in time with specific and direct advice.

6. CONCLUSIONS

After reviewing the application in its entirety, the biggest issue we have found and are particularly set on trying to improve in the future is the response time from the back-end server. This continues to be an issue, as the back-end is quite an important piece of the application and the tasks it performs are quite resource intensive [14]. The obvious best fix for this issue is to invest energy into a better and more equipped back-end server, as right now the server we are using is a free render.com server. Possibly in the future we could switch over to a AWS EC2 server, which does take a bit more configuration, however, the results and response times would definitely improve [15]. Additionally, the 'play' feature that enables the user to play chess is only a local feature. In the future, we think it would be fun and exciting to add a multiplayer online feature and enable the user's to enter match-making and play chess online.

Throughout the planning and development process for the GrandMaster Opening's application, I have learned a lot about what it takes to create an extensive and quality mobile app. The experience was truly insightful and I have walked away learning a lot about developing.

REFERENCES

[1] Donovan, Logan. The Complete Guide to Chess: Master: Chess Tactics, Chess Openings and Chess Strategies. 2021.
[2] Lehana, Paras, et al. "Statistical analysis on result prediction in chess." International Journal of Information Engineering and Electronic Business 11.4 (2018): 25.
[3] Baxter, Jonathan, Andrew Tridgell, and Lex Weaver. "Learning to play chess using temporal differences." Machine learning 40 (2000): 243-263.
[4] Puddephatt, Antony, and Gary Alan Fine. "Chess as art, science, and sport." A Companion to Sport (2013): 390-404.
[5] Smith, Eric. Talking Chess: A Phenomenological Study of Strategy and Social Reasoning in Chess Players. Diss. Northcentral University, 2021.
[6] Nandy, Abhishek, and Manisha Biswas. Reinforcement Learning: With Open AI, TensorFlow and Keras Using Python. Apress, 2017.
[7] Boukhary, Shady, and Eduardo Colmenares. "A clean approach to flutter development through the flutter clean architecture package." 2019 international conference on computational science and computational intelligence (CSCI). IEEE, 2019.

[8] Snider, L. A., and S. E. Swedo. "PANDAS: current status and directions for research." Molecular psychiatry 9.10 (2004): 900-907.
[9] Ayers, Richard, Benjamin Livelsberger, and Barbara Guttman. Quick start guide for populating mobile test devices. No. NIST Special Publication (SP) 800-202 (Draft). National Institute of Standards and Technology, 2018.
[10] Le Bot, Jacques, et al. "DART: a software to analyse root system architecture and development from captured images." Plant and Soil 326 (2010): 261-273.
[11] Pan, Chandan, et al. "ChatGPT: A OpenAI platform for society 5.0." Doctoral Symposium on Human Centered Computing. Singapore: Springer Nature Singapore, 2023.
[12] Liu, Yi, et al. "Prompt Injection attack against LLM-integrated Applications." arXiv preprint arXiv:2306.05499 (2023).
[13] Wolfson, Mike, and Donn Felker. Android developer tools essentials: Android Studio to Zipalign. " O'Reilly Media, Inc.", 2013.
[14] Levene, Mark, and Judit Bar-Ilan. "Comparing typical opening move choices made by humans and chess engines." The Computer Journal 50.5 (2007): 567-573.
[15] Hoque, Masudul. Classification of Chess Games: An Exploration of Classifiers for Anomaly Detection in Chess. Minnesota State University, Mankato, 2021.