# GAME RECOMMENDATION MODEL

## INTRODUCTION-

Our recommendation model gathers data from a dataset containing user data and another one containing steam video game data. It takes a user id and a random game and tells whether a user would like that game or not.

The dataset used in this project was provided by professor McAuley and originates from research on Steam game recommendations. It includes user-level interactions, such as playtime and preferences, alongside game attributes like genres, tags, price, and sentiment. This dataset is uniquely valuable for exploring gaming-specific user behavior and recommendation systems due to its rich combination of user actions and detailed item metadata.

Relevant studies include Kang and McAuley (2018), which proposed SASRec, a self-attention-based sequential recommendation model that leverages user behavior patterns, and Wan and McAuley (2018), which explored monotonic behavior chains to improve recommendations using implicit signals like playtime. Pathak et al. (2017) demonstrated how game metadata and user preferences can enhance personalized bundle recommendations on Steam. These studies emphasize the importance of metadata, sequential patterns, and implicit signals, which our project also leverages by incorporating features such as playtime, sentiment, popularity, genres, specs, and tags.

In addition to the Steam dataset, the Amazon Product Review dataset has been widely studied for user-item interaction and recommendation systems. It contains user reviews, product metadata, and implicit feedback, such as purchase histories, enabling researchers to analyze user preferences and predict interactions. While the Amazon dataset spans various product categories, the Steam dataset uniquely focuses on gaming-specific behavior, offering richer sequential patterns and metadata tailored to the domain.

State-of-the-art methods for studying such data include attention-based models like SASRec, which capture sequential patterns in user interactions, and collaborative filtering techniques that leverage latent user-item features. Deep learning models, such as recurrent neural networks, are also commonly used to analyze implicit feedback and textual metadata in datasets like Amazon's. Our project aligns with these methods by leveraging both user preferences and detailed game attributes.

While we begin with a simplified version of Jaccard similarity for the baseline, we expand to a more advanced model that incorporates additional weighted features such

as sentiment analysis from user reviews, two-week popularity (how many users played the game recently), genre similarity, weight similarity (capturing how closely a user's gameplay habits align with a game's characteristics), and game specifications like whether it's single-player or multiplayer. Unlike prior studies that employ complex attention-based or behavior-chain models, our approach focuses on interpretable and effective methods. Our findings align with existing literature in identifying sentiment, playtime, genres, and tags as critical predictors of user engagement, which we use to decide whether a user "likes" a game or not, while demonstrating significant improvement over the baseline through these feature-rich techniques.

**DATA-**

We're working with two datasets to build our recommendation system. The first dataset contains user information, including their user ID and a list of games they've played. For each game, the dataset includes details like the game's ID (item_id), name, total playtime (playtime_forever), and recent playtime over the last two weeks (playtime_2weeks). We'll use playtime_forever to measure the overall popularity of a game and playtime_2weeks to gauge its recent popularity.

The second dataset focuses on game metadata. It includes details like the game ID, name, genres, tags, price, discounted price (if applicable), specs, and sentiment data (if available). Our approach involves cross-referencing the user-game interaction data from the first dataset with the game metadata from the second dataset. This allows us to analyze key factors such as the popularity of certain games and genres, pricing trends, specs, tags, and user sentiment.

For our exploratory analysis, we found that we can merge these two datasets using the item_id from the user-game interaction data and the id from the game metadata dataset. This gives us a comprehensive view of user behavior alongside game attributes. The datasets include 80,000 unique users and 32,000 unique games. Through this analysis, we discovered some interesting patterns that will shape our recommendation system:

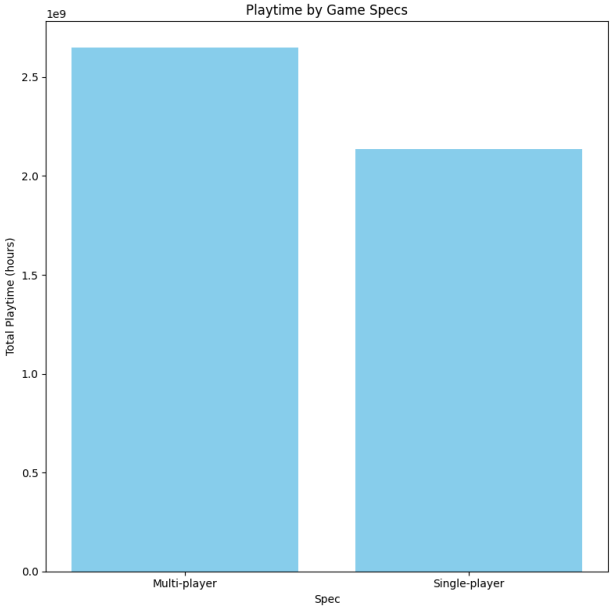
1. **Free-to-play games** dominate user engagement, while games with reasonable pricing also show strong performance. It was shown that users are more willing to play games that don't cost money up front. This tendency is significant since it implies that when forecasting user preferences, free-to-play games ought to be

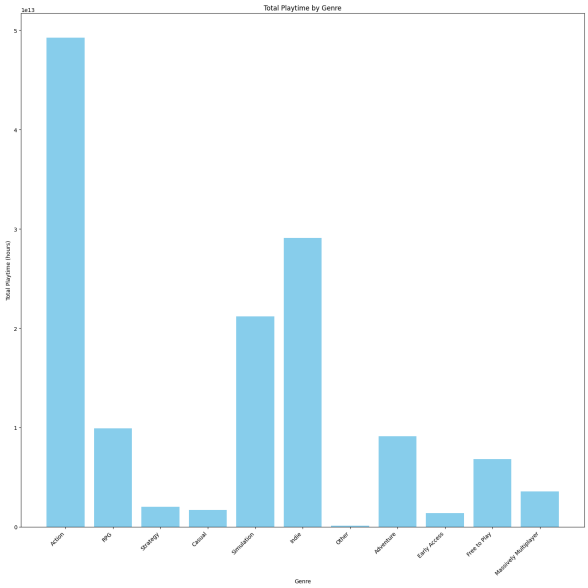
prioritized more. Knowing how well-liked these games are can assist our model in detecting a user's propensity to try out several games without committing to any one of them, which may be a crucial consideration when generating suggestions.

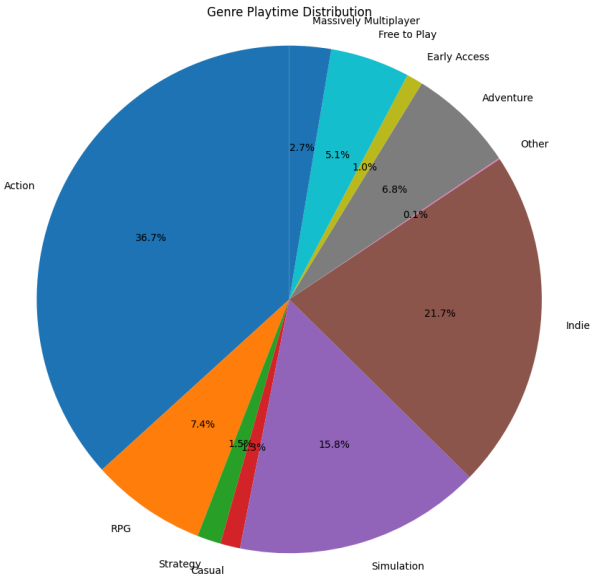
2. Popular games are mostly from the **action** and **indie** genres. This implies that when anticipating a user's preferences, our recommendation system ought to concentrate on these genres, particularly if the user has demonstrated a prior interest in playing these kinds of games. Based on the user's gaming behaviors and these genre preferences, the system can suggest related games that are more likely to appeal to them.
3. The playtime distribution is **highly skewed**, with a small group of users playing for significantly longer hours, which could distort the overall playtime metrics. Because these users may have a disproportionate impact on the model, this pattern raises the possibility of bias in the recommendations. We must make sure the model doesn't unduly favor these outliers in order to remedy this. This skewness can be lessened and the system can provide more balanced recommendations that are applicable to a wider spectrum of users by implementing weighted averages or normalization.
4. The amount of time in each genre a user spends. Users are more likely to like games that fit with their favorite genres. Consequently, the system ought to take into account both the total number of hours spent in a specific genre and the total range of genres a user interacts with. This will enable the model to recommend games from well-known genres or even present brand-new genres that fit well with the user's preexisting preferences.
5. Game Specifications are another important feature to watch out for. Games that fit the specifications of their system, whether in terms of performance or graphics, are more likely to be suggested to certain consumers. The user's experience can be improved by games that need system specs that are comparable to those they are used to. Specs should be included in the model even if they were determined to have less of an impact than playtime or genre, especially for users whose tastes are more in line with technical aspects or games with greater performance.

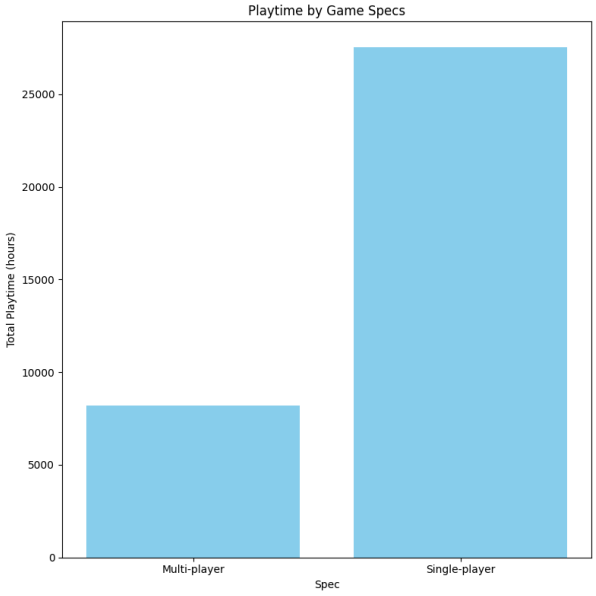
These findings provide critical insights and a solid starting point for designing our recommendation system. By leveraging the user-game interactions and game metadata, we

aim to create a system that accurately predicts whether a user will play a specific game.



## MODELS

We're using Steam's video game review data and pairing it with user and game information to create a recommendation system. The goal is to predict whether a user will play a specific game. For each (user, game) pair, the model gives a binary prediction: 0 for "not play" and 1 for "play," based on a probability score between 0 and 1. To see how well the model performs, we'll evaluate it using a train-test split and metrics like accuracy and mean squared error (MSE).

We've built two models for this. The first is a **baseline model** that's straightforward and uses features like the game's genre to measure similarity. The second model is a more **advanced model** that incorporates additional weighted features, including sentiment analysis from user reviews, two-week popularity (how many users played the game recently), genre similarity, and weight similarity (capturing how closely a user's gameplay habits align with the game's characteristics). These extra details help the advanced model make smarter, more accurate predictions about what games a user might enjoy.

To make it easier for the models to work with the data, we've organized it using defaultdict structs. This lets us neatly structure everything so it's quick and easy to reference during the modeling process. Below, you'll see how we've set up the data to build the foundation for the recommendation system.

```
gamesPerUser = defaultdict(list) #👍
hoursPerGame = {} #mostpopular all time
hoursPerGame2weeks = {} #mostpopular in the past 2 weeks
genrePerGamePerUser = defaultdict(list)
tagsPerGamePerUser = defaultdict(list)
hoursPerGamePerUser = defaultdict(list)##List of sets users: (game, hours) #👍
specsPerGamePerUser = defaultdict(list)
pricePerGamePerUser = defaultdict(float)
sentimentPerGameperUser = defaultdict(list) ##List of sets users: (game, sentiment)
hoursPerGenrePerUser = defaultdict(lambda: defaultdict(int))

positive_samples = []
negative_samples = []

# To generate negative samples, we need a set of all user-item interactions
all_items = set(item['item_name'] for user in userTrain for item in user['items'])
interaction_set = set()  # Keeps track of all user-item interactions
```

```
games = []
genreperGame = {}
specsperGame = {}
tagsperGame = {}
sentimentperGame = {}
priceperGame = {}
```

To ensure accurate results, we first split the database into 80%/20% split for each user's games. This allowed us to use 80% of the games each user played as training data, and the other 20% as testing data. The way we tested our model was by generating some negative samples and positive samples and combining them for the test case. We then gave the positive and negative samples labels respectively 1 and 0. We then compared the test case labels with the predictions our model spit out. We then cross referenced them and checked the accuracy of the model.

In addition to that, we had to "normalise" our data to get rid of null values to keep data consistent. For example, some games in game metadata don't have information stored. Some data have missing genre, name, or other features that were used in the model. To make data consistent, we had to deal with the null values and then make our prediction model accordingly. To do so, we store data in the structs mentioned above while dealing with the null cases or empty values. We don't include game data in which the game name is missing. Code implementation mentioned below.

Then, we made the baseline model. The baseline model looks at the amount of common genres the recommended game has with the games played by the user. It is a variation of Jaccard similarity which simplifies searching where we calculate the amount of common genres and find the ratio of commonality of the recommended game with each game the user has played. After that, we take the average of the ratios and predict whether a user will like the game or not based on whether it is above or below the threshold, which in this case was 0.5. Its code implementation is below.

```
games = []
genreperGame = {}
specsperGame = {}
tagsperGame = {}
sentimentperGame = {}
priceperGame = {}
for i in gameTrain:
  if 'app_name' in i:
    games.append(i['app_name'])
    if 'genres' in i:
      genreperGame[i['app_name']] = i['genres']
    else:
      genreperGame[i['app_name']] = []
    if 'specs' in i:
      specsperGame[i['app_name']] = i['specs']
    else:
      specsperGame[i['app_name']] = []
    if 'tags' in i:
      tagsperGame[i['app_name']] = i['tags']
    else:
      tagsperGame[i['app_name']] = []
    if 'sentiment' in i:
      sentimentperGame[i['app_name']] = i['sentiment']
    else:
      sentimentperGame[i['app_name']] = []
    if 'price' in i:
      priceperGame[i['app_name']] = i['price']
    else:
      priceperGame[i['app_name']] = []
```

```
[13] ### Baseline predictions (1) -
     def base_prediction(user, game):
       usergenres_per_game = genrePerGamePerUser[user]
       game_genre = genreperGame[game]
       if game_genre == []:
         return 0
       acc = 0
       sim_count = 0
       acc_counts = []

       for game,gamesGenre in usergenres_per_game:
         count = 0
         for genre in game_genre:
           if genre in gamesGenre:
             count += 1
         acc_counts.append(count/len(game_genre))
       if len(acc_counts) == 0:
         return 0
       acc = sum(acc_counts) / len(acc_counts)
       if acc > 0.5:
         return 1
       return 0
```

To make our recommendation system more effective and personalized, we enhanced the baseline model by incorporating additional features that provide deeper insights into user preferences. One of the major upgrades was the inclusion of game specifications, such as whether the game is single-player or multiplayer, along with the time a user spent playing a game. Additionally, we integrated two-week popularity, which measures how many users recently played a game, and sentiment analysis derived from user reviews to better understand how positively a game is perceived. These enhancements move beyond simplistic metrics like total playtime and focus on the context of the gaming experience and user engagement trends.

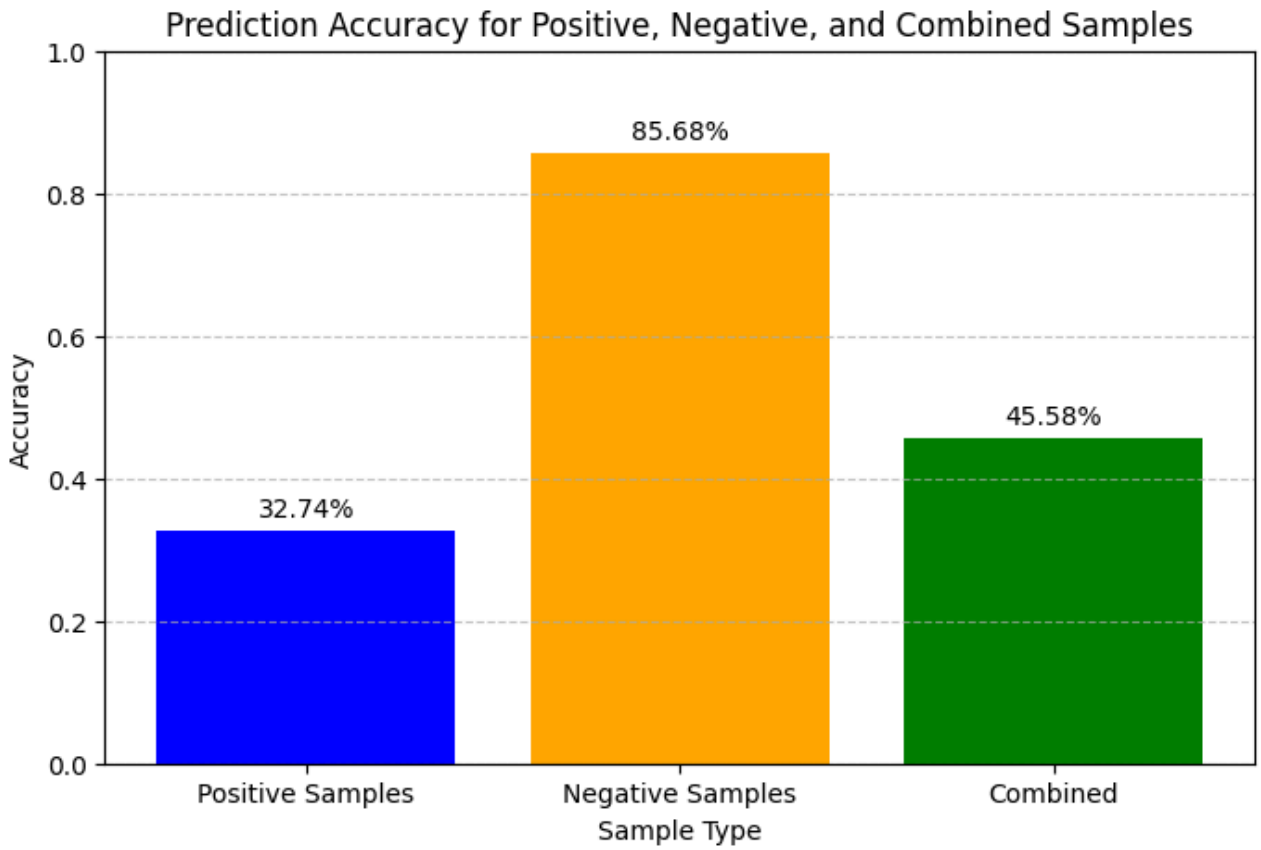
For example, story-driven games often have shorter completion times compared to open-world or multiplayer games. However, a shorter playtime doesn't necessarily mean the user didn't enjoy the game—it might have been a highly satisfying and memorable experience. Similarly, games with a high two-week popularity might indicate trending titles that are likely to attract users due to current interest, while sentiment analysis ensures that the overall perception of the game is taken into account. By incorporating these nuanced features, our model gains a more comprehensive understanding of user preferences and game characteristics, resulting in smarter and more personalized recommendations.

To address this, we added these game specs as a key feature to make the recommendations more nuanced and accurate. The advanced model uses a similarity function that compares the features of a target game—like its genre, mode, and average playtime—with the user's gaming history. This function calculates a percentage similarity between the target game and the user's preferences. The model then evaluates whether the game aligns with the user's interests by analyzing both genres and specs (including playtime) and checks if the similarity scores exceed a certain threshold. If they do, the system predicts that the user would enjoy the recommended game.

This approach allows us to factor in the unique characteristics of different games while avoiding common pitfalls, like undervaluing shorter games that users may have loved. By focusing on both the objective and subjective aspects of gaming preferences, the system delivers recommendations that feel more tailored and meaningful to each user. Below is the code that implements this enhanced recommendation logic.

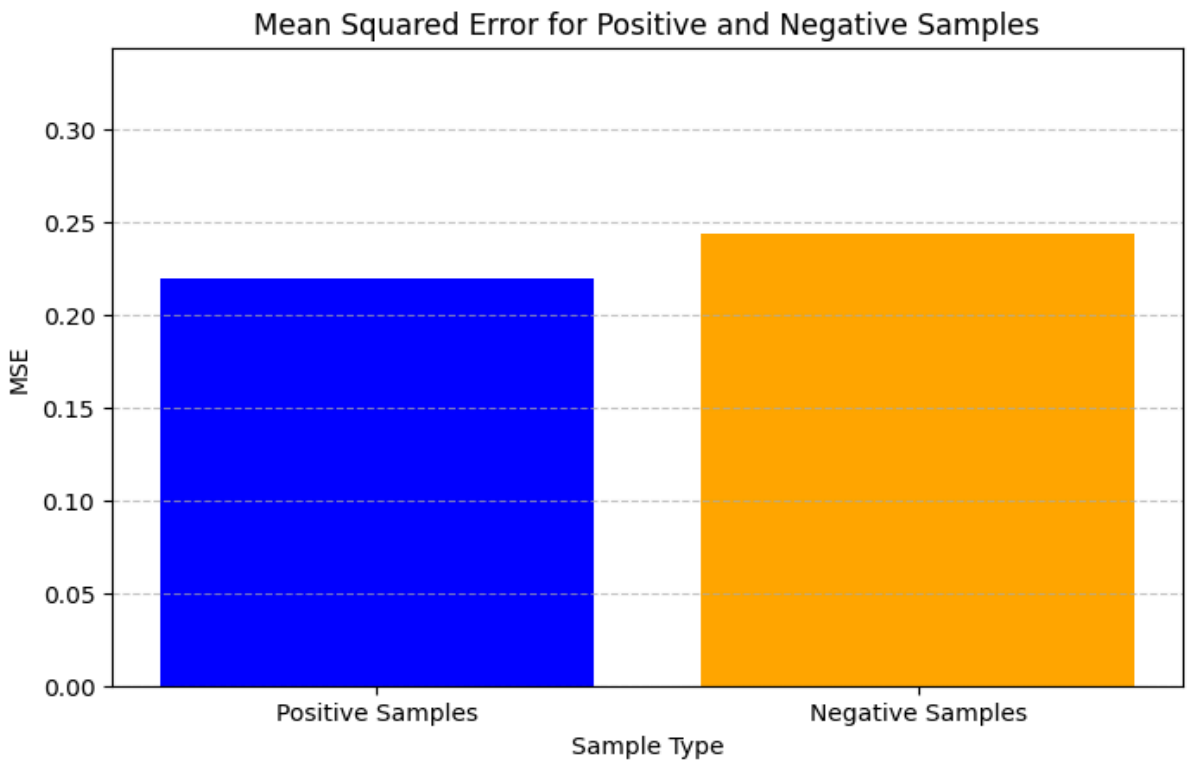
```python
def base_prediction3(user, game, genre_threshold=0.3, spec_threshold=0.3, genre_hours_threshold=40, min_matching_genres=2):
    # Retrieve user data
    usergenres_per_game = genrePerGamePerUser.get(user, [])
    userspecs_per_game = specsPerGamePerUser.get(user, [])

    # Retrieve game data
    game_genre = genreperGame.get(game, [])
    game_specs = specsperGame.get(game, [])
    game_sentiment = sentimentperGame.get(game, "")
    top_1200_games = list(hoursPerGame2weeks.keys())[:1200]
    #top_30_games_of_all_time = list(hoursPerGame2weeks.keys())[:20]

    # Ensure sentiment is a string, in case it's a list
    if isinstance(game_sentiment, list):
        game_sentiment = game_sentiment[0] if game_sentiment else ""

    def calculate_similarity(target_features, user_features):
        acc_counts = []
        for _, features in user_features:
            count = sum(1 for feature in target_features if feature in features)
            acc_counts.append(count / len(target_features) if target_features else 0)
        return sum(acc_counts) / len(acc_counts) if acc_counts else 0



    # Function to calculate weighted similarity based on playtime
    def calculate_weighted_similarity(game_genres, user_genres, user_hours):
        weighted_sim = 0
        total_hours = sum(user_hours.values())
        for genre in game_genres:
            if genre in user_genres:
                weighted_sim += (user_hours.get(genre, 0) / total_hours)  # Weight by playtime proportion
        return weighted_sim

    # Check if game has sentiment information
    def check_sentiment(sentiment):
        # Define sentiment categories
        positive_sentiments = ["Very Positive", "Mostly Positive", "Overwhelmingly Positive"]
        if sentiment in positive_sentiments:
            return True
        return False

    # If game genres exist, calculate similarity based on genres
    if game_genre:
        matching_genres = [genre for genre in game_genre if genre in hoursPerGenrePerUser[user] and hoursPerGenrePerUser[user][genre] > genre_hours_threshold]
        genre_sim = calculate_similarity(game_genre, usergenres_per_game)
        weighted_sim = calculate_weighted_similarity(game_genre, usergenres_per_game, hoursPerGenrePerUser[user])
        if game in top_1200_games:
            return 1
        if genre_sim > genre_threshold or weighted_sim > 0.20:
            if check_sentiment(game_sentiment):  # Check sentiment
                return 1  # Recommended based on genre similarity and sentiment
            else:
                return 0  # Not recommended based on sentiment

    # If game has specs, calculate similarity based on specs
    if game_specs:
        spec_sim = calculate_similarity(game_specs, userspecs_per_game)
        if spec_sim > spec_threshold:
            if check_sentiment(game_sentiment):  # Check sentiment
                return 1  # Recommended based on specs similarity and sentiment
            else:
                return 0  # Not recommended based on sentiment

    # If neither genre nor specs match, return 0 (not recommended)
    return 0
```

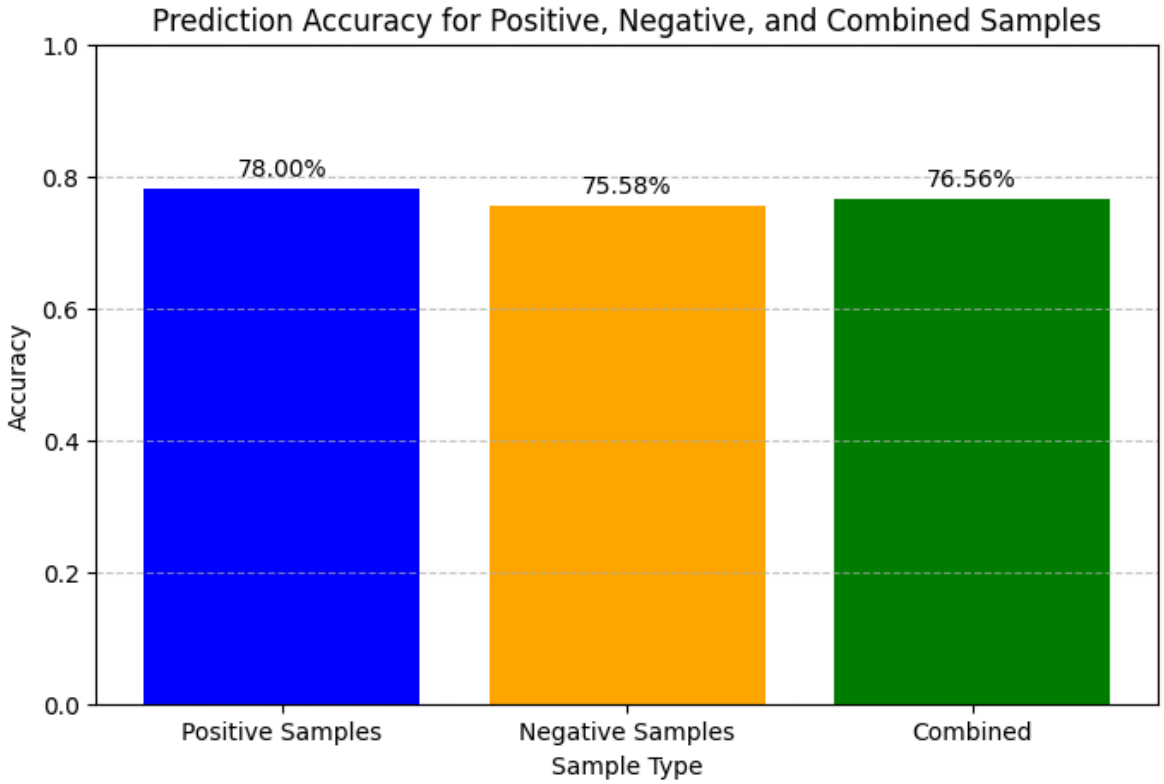
## RESULTS AND DISCUSSION-




(Baseline 1 model)

(Advanced Model)




**Findings and Recommendations**

The proposed model shows that it can accurately forecast user preferences by integrating a number of variables, including game specs, genres, popularity, sentiment, and playing hours. The model achieves more accuracy than more straightforward baselines that rely on just one of these criteria, such as genre or specifications alone, by combining these features with empirically set thresholds. In particular, games that fit users' playtime habits and genre preferences have higher recall in the model, which reduces the number of false negatives in the suggestions. This suggests that a hybrid model that makes use of several user behavior aspects can better describe user preferences and achieve better results.

The model's user-centric design is what makes these results significant. The approach can offer more individualized recommendations by directly integrating quantifiable user behaviors, such as the number of hours spent listening to particular genres. The increased accuracy implies that a more robust system results from combining several factors rather than depending just on one feature. This strategy might be used in other fields where a variety of factors affect judgment.

**Features**

The model's feature representations produced a range of outcomes. The mix of game hours and genres was really effective. The model matched user activity and engagement patterns by taking into account the quantity of matching genres and establishing a threshold of more than 40 hours per genre. Users' preferred gaming genres and the level of involvement with those genres were both captured by these attributes. However, although still helpful, game specifications did not have as much of an impact on the accuracy of the model. Particularly for games where the specifications did not clearly connect with what people tended to enjoy, the fixed criterion of 0.3 for specification accuracy sometimes failed to capture the subtleties of user preferences.

**Threshold**

When deciding if a game is a good fit for a user, the model's parameters—such as the thresholds for genre correctness (0.3), specification accuracy (0.3), and gaming hours (40 hours), sentiment, minimum matching genres, and 2 week popularity—act as decision limits. By reflecting user behavior, these thresholds make sure that a game is only suggested when it satisfies specific requirements. The "2-genre match" condition, for example, highlights the importance of a significant genre overlap when producing recommendations. In a similar vein, only genres with high user interaction are taken into account throughout the suggestion process thanks to the playtime hours threshold. These settings are intended to ensure that the model offers insightful suggestions without being overly restricted by striking a balance between specificity and generalization.

**Successes**

The model's hybrid nature, which blends various user preferences (genres and specs) and gameplay behaviors (hours spent per genre), is responsible for its success. Compared to single-feature models, the model was able to make recommendations that were more accurate by taking into account a variety of features of a user's profile. The model's robustness was further enhanced by the fallback technique, which ensures that suggestions can still be generated with partial data by depending on specifications in the event that genre data is missing.

**Why Other Models Failed**

Other models, including genre-only or specification-only models, didn't work since they weren't flexible enough to account for the entire spectrum of user preferences. While a spec-only model disregarded the user's favored genres, a genre-only model did not take into consideration how much time a user spends in a specific genre. Furthermore, the fixed thresholds applied to both genres and requirements might not be the best option for everyone. They might perform less than optimally if they are unable to generalize well across various user groups or datasets. Additionally, if requirements are not a good indicator of user preferences, using them as a fallback option may degrade performance. The adaptability and efficacy of the model could be further increased by adjusting these criteria and adding feature weighting.

## CONCLUSION-

In this project, we created a recommendation system that estimates a user's likelihood of enjoying a particular game based on a number of variables, such as playtime, genres, game parameters, and other game features. Unlike simpler models that concentrate on a single aspect, the model effectively incorporates many features, providing a more thorough approach. Through the integration of user behaviors, such as the amount of time spent on particular genres, with game attributes, such as genres and specifications, sentiment, and popularity, the model enhanced memory and produced more accurate predictions, lowering the possibility of false negative recommendations.

Its success was largely due to player preferences and gameplay patterns. A more accurate recommendation system was produced by combining genre preferences with playtime thresholds, looking at popularity and sentiment, and the model's reliability was increased by the fallback method, which uses specifications in the event that genre data is unavailable. But compared to genres and playtime, we discovered that game specs had less of an effect, indicating that although they can guide suggestions, they might not be as important in determining user preferences.

The importance of flexibility in capturing a range of user choices was highlighted by comparing with alternative models, such as genre-only or spec-only models. These more straightforward models were less flexible to various user profiles and failed because they failed to take into consideration the entire range of user activities. By adjusting the thresholds and adding feature weighting, the model's performance was further enhanced.

Everything being considered, our recommendation model shows the benefits of combining various game features and user behavior features, which makes it an appealing option for practical uses in game recommendations. The model provides a more individualized and efficient method of predicting user preferences by taking into consideration both obvious features (such as genres) and implied indicates (such as playtime). For even more precise predictions, future developments might concentrate on enhancing feature importance and modifying decision thresholds.

**REFERENCES-**

**Self-attentive sequential recommendation**
Wang-Cheng Kang, Julian McAuley
*ICDM*, 2018

**Item recommendation on monotonic behavior chains**
Mengting Wan, Julian McAuley
*RecSys*, 2018

**Generating and personalizing bundle recommendations on Steam**
Apurva Pathak, Kshitiz Gupta, Julian McAuley
*SIGIR*, 2017

**Justifying recommendations using distantly-labeled reviews and fined-grained aspects**
Jianmo Ni, Jiacheng Li, Julian McAuley
*EMNLP*, 2019