Implementing a Movie Recommender System

1st Jay Patel
Stevens Institute of Technology
CPE 695-WN
Old Bridge, USA
jpatel11@stevens.edu

2nd Kishan Shah
Stevens Institute of Technology
CPE 695-WN
Monroe, USA
kshah17@stevens.edu

3rd Khalid Isahak
Stevens Institute of Technology
CPE 695-WN
Englewood, USA
kisahak1@stevens.edu

***Abstract*—We are developing a movie recommendation system using the MovieLens 20M Dataset, which will be implemented using various algorithms such as collaborative filtering, content filtering, and matrix factorization. Collaborative filtering will predict user interests based on the preferences of similar users, while content filtering will recommend movies based on the movie attributes, like genre. Matrix factorization specializes in sparse matrices and is an extension of collaborative and content filtering. By leveraging different filtering algorithms and matrix factorization, our system aims to provide robust and accurate movie suggestions, enhancing the user experience.**

I. Introduction

In our project, we are going to implement a recommendation system for movies. The way we envision this working is as follows: Suppose Person X enjoyed movie A. Based on some parameters from movie A, we will ask our algorithm to generate suggested movies. In return, we expect to get an accurate list of movies similar to movie A. Similar to how Netflix generates suggested movies based on past viewing history and Spotify generates suggested music based on past listening history, we aim to minimize the issue where a viewer is stuck on what to watch next or where a viewer has no idea where to start and wants to search for movies similar to a specific title they have in mind.

In our approach to building our recommender system, we will utilize the MovieLens 20M Dataset from GroupLens. This dataset has 20 million ratings applied to 27,000 movies by over 135,000 users. The ratings range from 1 to 5 and include the timestamp of the rating, as well as other metadata like the movie genre and title.

We will implement this recommender system using three different approaches. The first approach is known as "collaborative filtering." In collaborative filtering, we make predictions about a user's interests based on the preferences of other users. For example, if user X likes movie A and movie B, and user Y likes only movie A, we would recommend movie B to user Y since user X also liked movie A and enjoyed movie B. Essentially, with collaborative filtering, we assume that since users have agreed in the past, they will also agree in the future. Collaborative filtering can be of two types: user-based or item-based. In user-based collaborative filtering, we make recommendations based on similarities between users. To compute the similarity between users, we will find the cosine similarity. After computing the similarity measure, we will find users similar to each other and recommend items that similar users have liked in the past, which the target user has not interacted with. In item-based collaborative filtering, we find the similarity between movies themselves. The steps are similar to user-based collaborative filtering: first, we compute the similarity between movies using cosine similarity based on features of the movies. Then, we find similar movies and ultimately suggest movies similar to the ones that the target user has previously liked.

The second approach is known as "content filtering." Content filtering is similar in some ways to collaborative filtering, but it uses a representation of a user's preferences as dictated by their past behavior, from likes to searches, to recommend content. In our case, we will use data like a user's searches, liked movies, and recently watched movies. For example, if a user searched for movie A and watched movie B, and another user watched both movies and liked movie C, we would recommend movie C to the searcher.

Finally, our third approach will utilize matrix factorization. Matrix factorization addresses the sparsity, often found in datasets to be funneled into a recommender system by decomposing the large, sparse user-item interaction matrix into two lower-dimensional matrices: one representing users and the other representing items. These matrices capture latent factors—hidden features that explain the underlying relationships between users and movies. For example, these latent factors might correspond to abstract concepts that distinct users may share, like a preference for certain genres, directors, or movie styles.

Altogether, our approach is focused on utilizing a filtering mechanism to find similarities between various users and recommend films to a user. By using different filtering algorithms, we aim to create a robust system to recommend movies similar to the ones in a user's search criteria. This system will continue to learn and feed information to help recommend movies to other users. Through our use of collaborative filtering, we have been able to witness our algorithm accurately suggest movies of similar genres based on users' search criteria. We specifically focused on using only users' ratings to generate search results for other users.

Throughout various popular applications like Netflix, Spotify, and even Facebook, recommendation systems have been prevalent. Movie recommendation systems enhance user experience using methodologies like collaborative filtering (user-based and item-based) and content-based filtering. Recent advancements include hybrid models combining these approaches, matrix factorization techniques like SVD, and deep learning methods such as neural collaborative filtering and autoencoders. Our project aims to build on these methods using the MovieLens dataset to create a tailored recommendation

system. Ultimately, we believe this recommender system can perform better than existing solutions because of our use of different hybrid approaches, which will combine the strengths of different filtering types.

II. Related Work

Recommendation systems play a crucial role in enhancing user experiences across various domains, such as movies, books, and products. These systems are typically categorized into three main types: content-based filtering, collaborative filtering, and hybrid approaches. Content-based filtering focuses on recommending items similar to those a user has shown interest in, based on the features of the items themselves. For example, if a user enjoys action movies starring a particular actor, the system will suggest other action movies featuring that actor. This approach relies on analyzing item attributes such as genre, cast, and director to make recommendations. While content-based filtering doesn't require user interaction data and scales well, it may be limited by the scope of its feature representation and lacks the ability to incorporate broader user preferences.

On the other hand, collaborative filtering relies on user interaction data, such as ratings and preferences, to recommend items. This method can be further divided into user-based and item-based filtering. In user-based collaborative filtering, the system identifies users with similar tastes and suggests items they have liked. Item-based collaborative filtering, however, focuses on finding items similar to those the user has interacted with. While this approach can offer recommendations even without detailed item attributes, it may struggle with unique tastes and requires a significant amount of user interaction data to be effective.

Machine learning algorithms are pivotal in developing recommendation systems. Content-based filtering often utilizes techniques like cosine similarity to measure the closeness between item features, converting items into feature vectors and calculating the cosine of the angle between these vectors. Collaborative filtering may use matrix factorization techniques such as Singular Value Decomposition (SVD) to discover latent factors and predict user preferences. Evaluation of these algorithms involves metrics like Precision, Recall, F1 Score, and Mean Squared Error (MSE) for assessing recommendation accuracy and relevance. Techniques such as cross-validation and A/B testing can be employed to validate and fine-tune the performance of recommendation models.

In implementing a movie recommendation system using content-based filtering, the process involves several steps, including data cleaning, feature extraction, and model deployment. For instance, leveraging APIs like TMDB can provide additional movie details, which are then used to compute similarities between movies based on attributes like cast and genre. By focusing on the combination of these attributes, the system can recommend movies with a higher similarity score to those previously searched by the user.

III. Our Solution

A. Description of Dataset

For our recommendation system, we will utilize the MovieLens 20M Dataset from GroupLens. MovieLens itself has a web-based recommender system created by GroupLens research, a research lab in the Department of Computer Science and Engineering at the University of Minnesota, focusing on gathering data for recommender systems. This comprehensive dataset comprises 20 million ratings for 27,000 movies by over 135,000 users, providing a rich source of information for our algorithms.

Each user within the dataset has rated a movie on a scale from 1 to 5, with 1 being the worst and 5 being the best. There are two files being used within our algorithm creation, being the ratings dataset and the movies dataset. The ratings dataset has a column for user ID, which gives a unique ID to each user, a movie ID, which gives a unique ID to each movie, a rating, and a timestamp, which refers to the exact time the user provided a rating for the movie. Similarly, the movies dataset comprises of the movie ID, the title of the movie, and genres listed for the movie.

The ratings dataset has 25806 data points, and the mean of all these ratings is approximately 3.5, with a standard deviation of about 1.09. The minimum rating is 0.5, with a first quartile rating of 3.0, a second quartile rating of 3.5, a third quartile rating of 4.0, and a maximum rating of 5.0. This can suggest that the ratings typically tend to be more favorable of the movies. From the following distribution we can tell that a vast amount of the movies were rated very favorably, with ratings like 4.0, 3.0, and 5.0 being quite frequent, whereas ratings like 1.5, 0.5, and 1.0 were quite rare.

	userId	movieId	rating	timestamp
count	1.118343e+06	1.118343e+06	1.118343e+06	1.118343e+06
mean	3.767268e+03	8.636830e+03	3.529347e+00	1.095984e+09
std	2.165237e+03	1.909917e+04	1.051638e+00	1.595496e+08
min	1.000000e+00	1.000000e+00	5.000000e-01	1.169000e+03
25%	1.924000e+03	9.020000e+02	3.000000e+00	9.659625e+08
50%	3.750000e+03	2.143000e+03	4.000000e+00	1.097903e+09
75%	5.584000e+03	4.638000e+03	4.000000e+00	1.217274e+09
max	7.633000e+03	1.306420e+05	5.000000e+00	1.427764e+09

Fig. 1. Description of the data

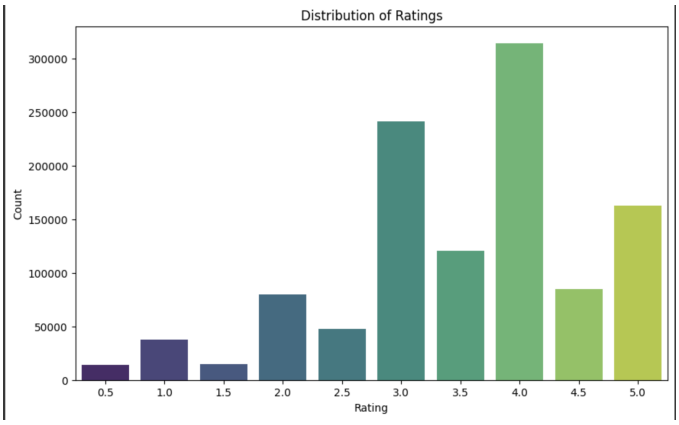

Fig. 2. Ratings distribution

During preprocessing, we will analyze the statistical properties of the raw data, such as the distribution of ratings, the shape of the data, and various other properties. Additionally, we will check for the existence of null values and remove all null values from the dataset. Furthermore, we will check for the existence of duplicates and remove any duplicates where it is necessary. For columns like genres, we will also format the strings so that the data is useful and easily accessible for our cause. Finally, we will break the ratings dataset into a train and test dataset, where we assure that one of each users' ratings is moved to the test set. We will utilize visualization techniques to help us identify and address issues like missing values, irrelevant features, and outliers. Feature engineering will include removing missing data, encoding categorical variables such as genres into numerical formats like a matrix, and utilize the normalized rating values to ensure consistency. These steps will enhance the dataset's quality, making it suitable for our collaborative filtering, content filtering, and matrix factorization.

B. Machine Learning Algorithms

For this movie recommendation system, two key machine learning algorithms will be used: Collaborative Filtering and Content-Based Filtering.

Collaborative Filtering suggests items to users by analyzing patterns and similarities in the preferences and behaviors of other users with similar tastes. This approach can be divided into user-based and item-based filtering. In user-based filtering, the system recommends items by identifying users who have similar preferences and suggesting items those users have liked. In item-based filtering, the system finds similarities between items and recommends items similar to those a user has liked. Collaborative filtering is particularly powerful because it does not require item metadata, relying purely on user behavior patterns to make recommendations, which makes it versatile and effective for large datasets.

Content-Based Filtering suggests items by analyzing the features of the items themselves and comparing them to the preferences of the user. For example, in a movie recommendation system, this could involve recommending movies with similar genres, directors, or actors to those the user has previously liked. This method excels in situations where user behavior data is sparse or nonexistent, as it relies on the inherent properties of the items rather than user interaction data.

To compute similarities in both content-based and collaborative filtering, Cosine Similarity is often used. Cosine similarity measures the cosine of the angle between two vectors in a multi-dimensional space. For two vectors A and B , cosine similarity is calculated as:

$$\text{Cosine Similarity} = \frac{\mathbf{A} \cdot \mathbf{B}}{|\mathbf{A}||\mathbf{B}|}$$

where the numerator is the dot product of the vectors, and the denominator is the magnitude of the vectors. A cosine similarity of 1 indicates that the items are identical, while a value of 0 indicates no similarity.

Another crucial technique in collaborative filtering is Matrix Factorization. This method involves decomposing a large user-item interaction matrix into lower-dimensional matrices that capture the latent factors underlying the data. A popular matrix factorization technique is Singular Value Decomposition (SVD), which factorizes the interaction matrix R into three matrices:

$$R = U\Sigma V^T$$

where U and V are orthogonal matrices, and Sigma is a diagonal matrix of singular values. These latent factors capture underlying patterns in user preferences and item characteristics, allowing for more accurate predictions and recommendations. Matrix factorization is particularly useful for handling large, sparse datasets where many interactions are missing, as it can predict missing values based on the latent features.

By leveraging these algorithms, the recommendation system can deliver accurate and personalized suggestions to users, making them well-suited for the MovieLens dataset.

Evaluating Recommender Systems: Evaluating the performance of recommender systems is particularly challenging due to several factors, including the cold start problem and the sparsity of data.

Cold Start Problem: The cold start problem occurs when the system struggles to make accurate recommendations for new users or items that have little to no interaction history. Since collaborative filtering relies heavily on user-item interactions, a new user or item with insufficient data can lead to poor recommendations. This is a significant challenge because the system doesn't have enough information to find similar users or items, which affects the accuracy of the recommendations.

Sparse Data Sets: Another challenge is the sparsity of the data. In most real-world scenarios, especially in large datasets like MovieLens, users have rated only a small subset of the available items. This means that the user-item interaction matrix is often sparse, with many missing entries. Such sparsity makes it difficult for collaborative filtering algorithms to identify meaningful patterns and relationships, leading to less accurate predictions.

One common evaluation metric for recommender systems is the Root Mean Squared Error (RMSE). RMSE measures the square root of the average squared differences between the predicted ratings and the actual ratings. It is calculated as:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

where y_i is the actual rating, y is the predicted rating, and n is the number of ratings. A lower RMSE indicates better predictive accuracy.

Leave-One-Out Split: To better evaluate the performance of our models, we employed a leave-one-out split rather than a traditional train-test split. This approach is particularly useful in recommendation systems where we aim to predict a user's

rating for a specific item while utilizing the user's existing ratings.

In this method, for each user, one rating is moved to the test set, and the remaining ratings form the training set. This ensures that we can evaluate the model's ability to predict a rating that is missing from the user's profile, closely simulating real-world scenarios.

C. Implementation Details

To implement our recommender system, we used Google Colab with an A100 GPU, which provided the necessary computational power for handling large datasets and complex algorithms. We began by importing the movies and ratings data from the MovieLens dataset in CSV format. The data was then loaded into Pandas DataFrames, which allowed for efficient data manipulation and analysis. For computing similarities and other model evaluations, we utilized tools from the Scikit-Learn library, such as cosine similarity.

Data Splitting and Model Evaluation: To test and evaluate our models, we implemented a leave-one-out split strategy. This method involves isolating a single rating for each user and using the remaining data to predict that rating. By doing so, we could closely simulate real-world scenarios where a recommendation system predicts a user's rating for an unseen item. The model's performance was then assessed by comparing the predicted ratings to the actual ratings, with the Root Mean Squared Error (RMSE) being our primary evaluation metric. Among the models tested, the one that handled sparse matrices most effectively achieved the lowest RMSE.

```
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import mean_squared_error

# Filter movies to only include those present in the training data
train_movie_ids = train_user_item_matrix.columns
filtered_movies = movies[movies['movieId'].isin(train_movie_ids)]

# Vectorize the genres for the filtered movies
count_vectorizer = CountVectorizer()
genre_matrix = count_vectorizer.fit_transform(filtered_movies['genres_str'])

# Compute the cosine similarity matrix
cosine_sim = cosine_similarity(genre_matrix)
```

Fig. 3. Cosine Similarity for Content Filtering

Regularization in Collaborative and Content Filtering: To prevent overfitting in our collaborative filtering model, we added regularization. Regularization helps by penalizing large weights in the model, thus preventing it from becoming too tailored to the training data. After testing various regularization parameters, we found that a parameter value of 0.1 provided the best balance, resulting in the lowest RMSE. This adjustment was crucial in improving the model's ability to generalize to new data.

```
from sklearn.metrics.pairwise import cosine_similarity

# Create the user-item matrix for the training data
train_user_item_matrix = train_data.pivot(index='userId', columns='movieId',
values='rating').fillna(0)

# Compute the cosine similarity matrix based on the training data
user_similarity = cosine_similarity(train_user_item_matrix)
user_similarity_df = pd.DataFrame(user_similarity, index=train_user_item_matrix.
index, columns=train_user_item_matrix.index)
```

Fig. 4. Cosine Similarity for Collaborative Filtering

Matrix Factorization and the Choice of K: In matrix factorization, a crucial parameter is the number of latent factors, denoted as K. These latent factors capture underlying patterns in the data, such as a user's preference for specific genres or movie characteristics. Selecting the optimal K is essential because it directly affects the model's ability to generalize and provide accurate recommendations.

We explored different values for K, starting with lower values such as 10 or 20, and gradually increasing to higher values like 50, 75, and 100. During this experimentation, we found that a lower K often led to underfitting, where the model was too simple to capture the complex relationships in the data. Conversely, a very high K could lead to overfitting, where the model became too tailored to the training data and performed poorly on unseen data.

```
from sklearn.metrics import mean_squared_error

predicted_ratings = []
actual_ratings = []

for _, row in test_data.iterrows():
    user = row['userId']
    movie = row['movieId']
    actual_rating = row['rating']

    if user in train_user_item_matrix.index and movie in train_user_item_matrix.
    columns:
        predicted_rating = predicted_ratings_df.loc[user, movie]
        predicted_ratings.append(predicted_rating)
        actual_ratings.append(actual_rating)

# Calculate RMSE
rmse = np.sqrt(mean_squared_error(actual_ratings, predicted_ratings))
print(f"Overall RMSE for Content-Based Filtering: {rmse}")
```

```
Overall RMSE for Content-Based Filtering: 3.753447776456357
```

Fig. 5. Evaluation using RMSE for Content Filtering

K = 75 emerged as the optimal choice because it provided the best balance between these two extremes. At this value, the model was able to capture enough latent factors to represent the underlying structure in the data without overfitting. This balance was reflected in the RMSE, which was lowest when K was set to 75, indicating the most accurate predictions.

```
import numpy as np
from scipy.sparse.linalg import svds

# Convert the user-item matrix to a NumPy array
user_item_matrix_np = user_item_matrix.to_numpy()

# Perform SVD
U, sigma, Vt = svds(user_item_matrix_np, k=75)

# Convert sigma to a diagonal matrix
sigma = np.diag(sigma)

# Reconstruct the user-item matrix
predicted_ratings = np.dot(np.dot(U, sigma), Vt)
predicted_ratings_df = pd.DataFrame(predicted_ratings, columns=user_item_matrix.
columns)
```

Fig. 6. Performing SVD for Collaborative Filtering

IV. Comparison

After training and evaluating the models using the leave-one-out split, we obtained the following RMSE values:

- Collaborative Filtering (Cosine Similarity):
  Test RMSE: 3.0506743395327223

- Content-Based Filtering (Cosine Similarity): Overall RMSE for Content-Based Filtering: 3.753447776456357
- Collaborative Filtering (Matrix Factorization): Test RMSE: 2.3016415173002938

The results show that the Collaborative Filtering model using Matrix Factorization achieved the lowest RMSE, indicating the best predictive performance among the models evaluated. This is likely due to the ability of matrix factorization techniques to uncover latent factors that explain the underlying structure in the data, even when the data is sparse.

However, it is important to note that even though the RMSE is relatively high, the quality of the movie recommendations can still be quite good, particularly when recommending within specific genres. For example, when using collaborative filtering with cosine similarity, the recommendations for the movie "Toy Story (1995)" included:

```
13 print(get_recommendations('Toy Story (1995)'))

2209                                        Antz (1998)
3027                                Toy Story 2 (1999)
3663        Adventures of Rocky and Bullwinkle, The (2000)
3922                Emperor's New Groove, The (2000)
4790                            Monsters, Inc. (2001)
10114    DuckTales: The Movie - Treasure of the Lost La...
10987                                 Wild, The (2006)
11871                         Shrek the Third (2007)
13337                  Tale of Despereaux, The (2008)
18274    Asterix and the Vikings (Astérix et les Viking...
Name: title, dtype: object
```

Fig. 7. Results of Collaborative Filtering with cosine similarity

These recommendations are highly relevant to the genre of "Toy Story," indicating that the model effectively captures the genre-specific preferences of the user, even if the overall RMSE is not optimal.

V. Future Directions

Looking ahead, there are several avenues through which we can further enhance the performance of our movie recommendation system, particularly in the areas of collaborative filtering, content-based filtering, and matrix factorization. One promising direction is the exploration of advanced collaborative filtering techniques, such as neural collaborative filtering (NCF), which utilizes deep learning to capture more complex user-item interactions beyond the linear relationships identified by traditional methods. Incorporating this could lead to more sophisticated and accurate recommendations. Additionally, enhancing our content-based filtering approach by integrating natural language processing (NLP) techniques could allow for a more nuanced analysis of movie plots, reviews, and other textual data, therefore improving the system's ability to understand and predict user preferences. The use of image processing on movie posters or analyzing audio features from soundtracks could also add another layer of depth to content-based recommendations.

In the realm of matrix factorization, further refinement could involve exploring advanced techniques such as non-negative matrix factorization or probabilistic matrix factorization (PMF), which may offer better handling of data sparsity and improved recommendation accuracy. Regularization techniques could also be enhanced to prevent overfitting, allowing the model to generalize better to unseen data. A hybrid model could be built to further provide a more effective combination of algorithms, leading to better overall performance. Additionally, developing contextual hybrid models that incorporate external factors like user location, time, or social influences could result in more contextually relevant recommendations.

Addressing the cold start problem remains a critical challenge, particularly for new users or items. Future efforts could focus on implementing transfer learning models that leverage information from other domains or datasets, improving predictions in these scenarios. Active learning techniques, where user feedback is solicited to quickly gather data, could also be explored to mitigate the cold start issue. Another key area for future work is the implementation of real-time recommendation capabilities, which would significantly enhance user experience. By optimizing algorithms for real-time processing and integrating streaming data, the system could provide instant, personalized suggestions as users interact with the platform. This would likely involve leveraging distributed computing frameworks to handle large-scale data efficiently.

As the system becomes more complex, it will be important to focus on transparency and user trust. Integrating explainable AI techniques could provide users with clear explanations for why certain movies were recommended, thereby improving user satisfaction. Additionally, exploring deep learning models such as Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) for content-based filtering could further enhance the system's ability to automatically learn features from raw data, such as movie plots, reviews, or trailers.

By pursuing these future directions, we can continue to refine our movie recommendation system, making it more accurate, scalable, and user-friendly, ultimately delivering even more relevant and satisfying recommendations to users.

VI. Conclusion

While the recommender system performed reasonably well, there are several ways we could have improved our approach, starting with data preparation. One significant challenge was the sparsity of the data, which likely contributed to the higher RMSE values observed. To address this, we could have reduced the sparsity by removing movies and users who did not have enough ratings. This would have allowed the models to focus on more informative interactions, potentially improving the accuracy of the recommendations.

Another area for improvement could be the implementation of a hybrid method, which combines both collaborative and content-based filtering. By incorporating additional features beyond just genres, such as user demographics or movie metadata, we could enhance the quality of the recommendations. For collaborative filtering, considering other user attributes, like viewing history or location, might also have led to more personalized and accurate recommendations.

Moreover, we could have employed a broader range of evaluation criteria. While RMSE provides valuable insights into predictive accuracy, other metrics like Precision at K,

Recall, and F1-score could have given us a more comprehensive understanding of the system's performance, particularly in terms of its ability to rank relevant items highly.

In summary, by focusing on better data preparation, exploring hybrid methods, and using a wider array of evaluation metrics, we could have developed a more robust and effective recommender system.

REFERENCES

[1] Rao, N. K., Challa, N. P., Chakravarthi, S. S., & Ranjana, R. (2022). Movie Recommendation System using Machine Learning. In Proceedings of the 2022 4th International Conference on Inventive Research in Computing Applications (ICIRCA) (pp. 711-716). IEEE. doi: 10.1109/ICIRCA54612.2022.9985512.

[2] Wu, C.-S. M., Garg, D., & Bhandary, U. (2018). Movie Recommendation System Using Collaborative Filtering. In Proceedings of the 2018 IEEE 9th International Conference on Software Engineering and Service Science (ICSESS) (pp. 11-15). IEEE. doi: 10.1109/ICSESS.2018.8663822.

[3] Soni, N., Kumar, K., Sharma, A., Kukreja, S., & Yadav, A. (2021). Machine Learning Based Movie Recommendation System. In Proceedings of the 2021 IEEE 8th Uttar Pradesh Section International Conference on Electrical, Electronics and Computer Engineering (UPCON) (pp. 1-4). IEEE. doi: 10.1109/UPCON52273.2021.9667602.