Improving Recommendation Quality on YouTube and Netflix with Singular Value Decomposition

Haegen Quinston - 13523109[2]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13523109@std.stei.itb.ac.id, [2]haegenquinston@gmail.com

Abstract*—Recommendation systems have become an integral part of enhancing user experience across various platforms by providing personalized suggestions. Singular Value Decomposition (SVD), a matrix factorization technique, is widely adopted in collaborative filtering-based recommendation systems to address challenges such as scalability and cold start issues. This paper investigates the effectiveness of SVD as a standalone method for building recommendation systems. Testing was conducted across user activity clusters and movie popularity groups to evaluate the model's predictive accuracy through metrics such as Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE). The results indicate that while SVD captures latent relationships effectively, it struggles with sparsity and new user challenges, emphasizing the need for hybrid approaches and advanced techniques for robust recommendation quality.

***Keywords*—SVD, user preference, model, sparsity**

I. Introduction

Recommendation systems have become an integral part of our digital experiences, shaping the way we interact with various platforms and consume content. Across industries, these systems drive personalization, offering users tailored suggestions that enhance engagement and satisfaction. From e-commerce platforms guiding customers to relevant products, to streaming services like Netflix and Spotify curating content based on viewing and listening habits, recommendation systems are pivotal in meeting user expectations. Similarly, social media platforms use these tools to promote meaningful connections and trends, while the gaming industry leverages them to provide players with immersive and personalized in-game experiences. These applications not only improve user retention but also significantly contribute to the business success of these platforms.

Despite their success, recommendation systems face critical challenges, particularly in addressing scalability and the cold start problem. Scalability refers to the ability to handle massive datasets efficiently as the number of users and items grows. Platforms like YouTube and Netflix manage enormous datasets comprising millions of user interactions and thousands of content items, requiring algorithms that balance computational efficiency with accuracy. The cold start problem, on the other hand, arises when there is insufficient data for new users or items, making it difficult to provide accurate recommendations. Overcoming these challenges is essential to maintaining the relevance and performance of recommendation systems in dynamic environments.

Singular Value Decomposition (SVD), a matrix factorization technique, has been widely regarded as an effective approach for addressing challenges in recommendation systems. By reducing the dimensionality of sparse user-item interaction matrices, SVD captures latent relationships between users and items, enabling predictions in large-scale systems. However, this paper aims to test whether SVD alone is sufficient to form a robust recommendation system. Through a series of evaluations, it highlights the strengths of SVD in identifying patterns, while also exposing its limitations, such as data sparsity and cold start challenges, which impact its predictive accuracy.

II. Theoretical Foundations

A. Recommendation Systems

Recommendation systems are specialized information filtering tools that provide personalized suggestions to users by predicting items of interest based on user behavior, preferences, and past interactions. These systems are categorized into three main types: Content-Based Filtering, Collaborative Filtering, and Hybrid Models.

Content-Based Filtering analyzes item attributes, such as genres or tags, to recommend items similar to those a user has previously interacted with. While effective for personalization, it often lacks diversity in suggestions.

Hybrid Models address the limitations of individual approaches by combining methods like Content-Based and Collaborative Filtering. By leveraging the strengths of both, hybrid models enhance recommendation diversity and accuracy. These systems are commonly used in large-scale platforms like YouTube and Netflix, where personalization and scalability are critical.

Meanwhile, Collaborative Filtering (CF) is a widely adopted approach that predicts a user's interests by

analyzing preferences from multiple users. The core assumption is that if user A shares similar preferences with user B in one context, A is likely to agree with B in another.

CF is categorized into two types:

1. User-Based Collaborative Filtering: Recommends items that similar users have liked.
2. Item-Based Collaborative Filtering: Recommends items similar to those a user has liked.

A significant challenge in CF is the sparsity of the user-item interaction matrix, where many entries are missing due to the vast number of items and limited user interactions. This sparsity often hinders effective similarity computations between users or items. Each of these systems lays the foundation for advanced techniques like Singular Value Decomposition (SVD), which refines recommendation accuracy through matrix factorization.

B. Matrix Factorization

Matrix Factorization is a pivotal technique in collaborative filtering, widely utilized within recommendation systems to predict user preferences for items. The core idea involves decomposing a large user-item interaction matrix into the product of two lower-dimensional matrices, effectively capturing latent factors that represent underlying user interests and item characteristics.

Given a user-item interaction matrix R of dimensions $m \; x \; n$ (w m is the number of users and n is the number of items), MF approximates R as the product of two matrices [5]:

$$R \approx \mathrm{U} \; \mathrm{x} \; \mathrm{V}^{\mathrm{T}}$$

- U: An $m \; x \; k$ matrix where each row corresponds to a user's latent factors.
- V: An $n \; x \; k$ matrix where each row corresponds to an item's latent factors.
- k: The number of latent factors, typically much smaller than m or n, reducing the dimensionality of the data.

C. Singular Value Decomposition

Singular Value Decomposition (SVD) is a factorization of a certain matrix into 3 matrices. It has several algebraic properties and conveys important insights regarding linear transformations. SVD is widely used in various fields, such as data science and data compression, among others.

For a given matrix A of size $m \; x \; n$, SVD decomposes it into three matrices [4]:

$$A \; = \; U \sum V^{T}$$

The matrix U is an $m \; x \; m$ orthogonal matrix, where its columns are left singular vectors of A. U has a unique property, where all its columns are orthogonal ($U^{\mathrm{T}}U = I$) and the number of non-zero columns in U corresponds to the rank of A. U represents the row space of A in the transformed domain.

The matrix $\sum$ is an $m \; x \; n$ diagonal matrix with singular values (σ1, σ2, …) sorted in descending order along the diagonal. The singular values are the square roots of the eigenvalues of $A^{\mathrm{T}}A$. Singular values indicate the magnitude of the corresponding singular vectors and provide insight into the importance of each component.

Finally, V^{T} is the transpose of an $n \; x \; n$ orthogonal matrix, where V contains the right singular vectors of A. The vectors form an orthonormal basis for the row space of A. Each column in V^{T} can be interpreted as a 'direction' in the transformed space of the columns of A.

While SVD provides a mathematical framework to extract latent user and item factors, its performance is heavily influenced by the sparsity of the interaction matrix. High levels of missing data can lead to inaccurate predictions, as the decomposition relies on patterns in available ratings. Furthermore, SVD struggles with the cold start problem, where insufficient historical data for new users or items hinders its ability to generate meaningful recommendations.

D. Root Mean Squared Error & Mean Absolute Error

The RMSE (Root Mean Squared Error) is a widely used metric to measure the differences between predicted values and observed values. It places greater emphasis on larger errors, making it particularly sensitive to outliers.
For a given set of user-item pairs T, RMSE is calculated as follows:

$$RMSE = \sqrt{\frac{\sum_{(u,i)\in T}(r_{ui} - \hat{r}_{ui})^2}{|T|}}$$

Where:

- r_{ui} is the actual rating of user u for item i
- $\hat{r}_{ui}$ is the predicted rating of user u for item i

A lower RMSE value indicates better predictive accuracy, making it a critical metric for evaluating recommendation systems.

The MAE (Mean Absolute Error) measures the average absolute difference between predicted ratings and actual ratings. It is calculated using the same variables as RMSE:

$$MAE = \frac{\sum_{(u,i)\in T} |r_{ui} - \hat{r}_{ui}|}{|T|}$$

Unlike RMSE, MAE treats all errors equally and is less sensitive to outliers. A lower MAE value also signifies more accurate predictions, offering a complementary perspective on model performance. Together, RMSE and

MAE provide a comprehensive understanding of a recommendation system's accuracy.

III. Implementation & Analysis

A. Dataset

The original dataset utilized in this study is obtained from GroupLens, a research organization that provides publicly available datasets collected from the MovieLens platform. These datasets are widely used in recommendation system research due to their detailed user-item interaction records and metadata. The dataset is freely accessible to the public and can be downloaded here.

For this study, the original large dataset was utilized to ensure robust evaluation of the recommendation system. The dataset contains millions of user interactions, including ratings, user IDs, and movie metadata. Using a large dataset helps to reflect the real-world complexities and challenges of recommendation systems, such as scalability and sparsity. Although, for computational purposes this dataset is reduced to just 10000 users and 2000 movies to ensure the program doesn't lag. A train-test split (80/20) was applied to the data to evaluate the model's predictive performance.

movies.csv

This file contains metadata about movies, including their unique identifiers, titles, and genres. Below are its columns:

- movieId: A unique identifier for each movie.
- title: The name of the movie, including its release year.
- genres: A pipe-separated list of genres associated with the movie.

Table 3.1: Sample data from movies.csv

movieId	title	genres
1	Toy Story (1995)	Adventure
2	Jumanji (1995)	Adventure
3	Grumpier Old Men (1995)	Comedy
4	Waiting to Exhale (1995)	Comedy
5	Father of the Bride Part II	Comedy

ratings.csv

This file records the interactions between users and movies in the form of ratings. It includes the following columns:

- userId: A unique identifier for each user.
- movieId: A unique identifier for each movie (links to movies.csv).
- rating: The rating provided by the user for the movie, typically on a 0.5–5.0 scale.
- timestamp: A Unix timestamp indicating when the rating was submitted.

Table 3.2: Sample data from ratings.csv

movieId	movieId	rating	timestamp
1	1	4.0	964982703
1	3	4.0	964981247
2	1	5.0	964982224
2	4	2.0	964982931
3	1	4.0	964982400

From the dataset, histograms and heatmaps were generated to analyze the distribution of ratings and the sparsity of user-item interactions. These visualizations ensured that the data used in this experiment was representative of real-world challenges and highlighted potential biases or sparsity issues.

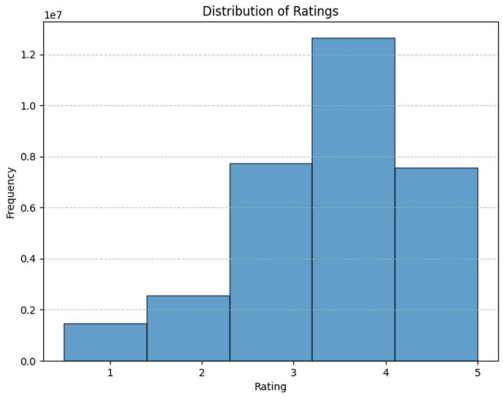

Fig 3.1: A histogram showing the distribution of ratings on the dataset

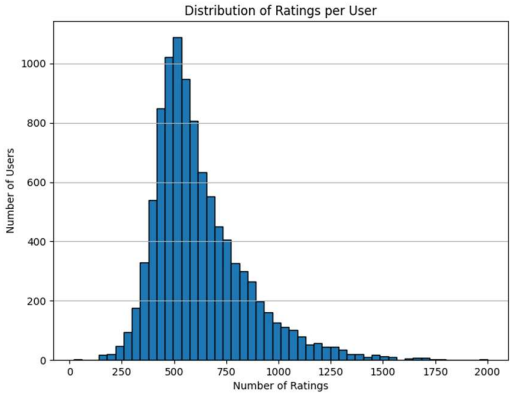

Fig 3.2: A bar chart showing the distribution of ratings per user

These visualizations demonstrated that the dataset provided a clear structure for testing, with histograms showcasing a well-defined distribution of rating scores and the number of ratings per user. The distribution analysis highlights the diversity of user interactions and rating behaviors, ensuring the dataset is representative for evaluating the model. Although, the cause for the majority of ratings per user being so high is because it is sorted for the top 10000 users with the most ratings.

This dataset reflects the kind of data typically utilized in recommendation systems for platforms like Netflix and

YouTube. In the case of YouTube, user interactions such as likes, dislikes, and subscriptions serve as analogs to the 'ratings' column, while videos take the place of movies, highlighting the adaptability of recommendation algorithms across different content types and user behaviors.

B. Recommendation System Model with SVD

The recommendation module made in Python is designed to implement a collaborative filtering-based recommendation system using Singular Value Decomposition (SVD). It begins by loading and preprocessing the dataset, filtering for the most active users and popular movies to optimize performance. The main functionality lies in the construction of a user-item interaction matrix, which is factorized using SVD into latent user and item feature matrices. This process identifies underlying patterns in user preferences and item characteristics, enabling predictions for unrated items. The module also includes functionality to generate personalized recommendations for a given user, ensuring that the recommended movies are not already watched.

a) Data Preprocessing

```
def load_data():
    ratings =
pd.read_csv('Recommendation_Syste
m_Modeling/test/large_ratings.csv
')
    movies =
pd.read_csv('Recommendation_Syste
m_Modeling/test/large_movies.csv'
)
    # Map userId and movieId to
sequential indices
    ratings['userId'] =
ratings['userId'].astype('categor
y').cat.codes
    ratings['movieId'] =
ratings['movieId'].astype('catego
ry').cat.codes

    # Split data into train and
test sets (80/20 split)
    test_fraction = 0.2
    test_indices =
np.random.choice(ratings.index,
size=int(len(ratings) *
test_fraction), replace=False)
    test_data =
ratings.loc[test_indices]
    train_data =
ratings.drop(test_indices)

    return train_data, test_data,
movies
```

Fig 3.3: The function for loading the data

The preprocessing process includes mapping user IDs and movie IDs to sequential indices and creating a sparse user-item matrix. An 80/20 train-test split was applied to ensure that the model could be evaluated on unseen data. This split ensures that the training set contains sufficient interactions to learn meaningful patterns while the test set provides an independent basis for evaluating prediction accuracy.

b) Matrix Factorization

```
def
build_recommendation_system(ratin
gs):
    user_item_matrix_sparse =
csr_matrix(
        (ratings['rating'],
(ratings['userId'],
ratings['movieId']))
    )

    # SVD
    U, sigma, Vt =
svds(user_item_matrix_sparse,
k=20)
    sigma = np.diag(sigma)

    # Predicted ratings
    predicted_ratings =
np.dot(np.dot(U, sigma), Vt)
    predicted_ratings_df =
pd.DataFrame(
        predicted_ratings,
        index=np.arange(user_item
_matrix_sparse.shape[0]),
        columns=np.arange(user_it
em_matrix_sparse.shape[1])
    )

    return predicted_ratings_df
```

Fig 3.4 The function for predicting ratings using SVD

This function constructs a sparse matrix from the `rating`, `userId`, and `movieId` columns, where the rows represent users, the columns represent movies, and the values correspond to user ratings. Singular Value Decomposition (SVD) is then applied to factorize the matrix into three components: U, which contains user latent features; $\sum$, the diagonal matrix of singular values representing the importance of these features; and V, which contains item latent features. Finally, the matrix is reconstructed by combining these components, and the resulting predictions are stored in a DataFrame for easy access and further analysis.

c) Prediction

```
def
recommend_movies_with_watched(use
r_id, predicted_ratings_df,
ratings, movies,
num_recommendations=10):
    # Check if the user exists
    if user_id not in
predicted_ratings_df.index:
        return f"User {user_id}
not found in the dataset.", []

    # Display watched movies
    watched_movies_ids =
ratings[ratings['userId'] ==
user_id]['movieId'].unique()
    watched_movies =
movies[movies['movieId'].isin(wat
ched_movies_ids)][['movieId',
'title']]

    # Generate recommendations
    user_ratings =
predicted_ratings_df.loc[user_id]
    recommendations_ids =
user_ratings[~user_ratings.index.
isin(watched_movies_ids)].sort_va
lues(ascending=False).head(num_re
commendations).index
    recommendations =
movies[movies['movieId'].isin(rec
ommendations_ids)][['movieId',
'title']]

    return watched_movies,
recommendations
```

Fig 3.5 The function for returning recommended movies

Personalized recommendations are generated to the user using this function. It reconstructs the matrix to predict ratings for missing user-item interactions. The function ensures that the film recommendations are not redundant with the user's previously watched content, but reflective of the user's ratings given.

While the dataset provides a more realistic evaluation of the recommendation system, it also introduces challenges such as computational complexity and scalability. These challenges were addressed by optimizing the SVD implementation and limiting the number of latent dimensions to balance accuracy and performance. Future studies could explore additional techniques, such as hybrid models or deep learning-based approaches, to further improve scalability and recommendation quality.

IV. Recommendation System Testing

A. Testing

Testing will be conducted using three distinct clusters, with RMSE and MAE calculated for each cluster type to evaluate the recommendation system's performance.

For the first sample, the data is segmented into three groups based on user activity levels: high, medium, and low activity users. This segmentation is achieved through the methods shown below:

```
user_activity = ratings['userId'].value_counts()
low_activity_users = user_activity[user_activity <= 100].index
medium_activity_users = user_activity[(user_activity > 100) & (user_activity <= 500)].index
high_activity_users = user_activity[user_activity > 500].index
```

Fig 4.1 Separating high, medium, and low activity users

The RMSE and the MAE is as follows:

```
Low Activity - RMSE: 2.8909, MAE: 2.5138
Medium Activity - RMSE: 2.2722, MAE: 2.0570
High Activity - RMSE: 1.9151, MAE: 1.6751
```

Fig 4.2 RMSE and MAE for the first sampling

The results indicate that Singular Value Decomposition (SVD) did not perform well across the tested datasets. According to established benchmarks, an RMSE in the range of 0.75–1.5 is generally considered good. However, the test results significantly exceeded this range, highlighting limitations in the SVD model's ability to handle the given data.

```
top_users = ratings['userId'].value_counts().nlargest(25000).index
top_movies = ratings['movieId'].value_counts().nlargest(5000).index
ratings = ratings[(ratings['userId'].isin(top_users)) & (ratings['movieId'].isin(top_movies))]

# Map userId and movieId to sequential indices
ratings['userId'] = ratings['userId'].astype('category').cat.codes
ratings['movieId'] = ratings['movieId'].astype('category').cat.codes

user_activity = ratings['userId'].value_counts()
low_activity_users = user_activity[user_activity <= 200].index
medium_activity_users = user_activity[(user_activity > 200) & (user_activity <= 500)].index
high_activity_users = user_activity[user_activity > 500].index
```

Fig 4.3 The setup for the 2nd sampling

In the second sample, the low-medium-high activity clusters were redefined, and the dataset size was expanded to include the top 25,000 user ratings and 5,000 movies.

```
Low Activity - RMSE: 3.1039, MAE: 2.8733
Medium Activity - RMSE: 2.6004, MAE: 2.3798
High Activity - RMSE: 2.2172, MAE: 1.9631
```

Fig 4.4 RMSE and MAE for the 2nd sampling

As expected, the RMSE for this configuration was even worse. Despite the increased number of movies, the system struggled due to the inherent sparsity of user ratings, which further impacted the model's performance.

```
# Cluster movies based on rating counts
movie_rating_counts = train_data['movieId'].value_counts()

# Define clusters
top_10_percent = movie_rating_counts.quantile(0.9)
moderate_40_percent = movie_rating_counts.quantile(0.5)

popular_movies = movie_rating_counts[movie_rating_counts >= top_10_percent].index
moderate_movies = movie_rating_counts[(movie_rating_counts < top_10_percent) & (movie_rating_counts >= moderate_40_percent)].index
niche_movies = movie_rating_counts[movie_rating_counts < moderate_40_percent].index

# Function to filter ratings by movie clusters
def filter_movies_by_cluster(cluster_movies, test_data):
    mask = test_data['movieId'].isin(cluster_movies)
    return test_data[mask]

# Calculate RMSE and MAE for each movie cluster
for cluster_name, cluster in [('Popular Movies', popular_movies),
                              ('Moderate Movies', moderate_movies),
                              ('Niche Movies', niche_movies)]:
    filtered_test_data = filter_movies_by_cluster(cluster, test_data)

    if filtered_test_data.empty:
        print(f"No test data for {cluster_name}. Skipping.")
        continue

    filtered_test_matrix = csr_matrix(
        (filtered_test_data['rating'], (filtered_test_data['userId'], filtered_test_data['movieId'])),
        shape=(predicted_ratings_df.shape[0], predicted_ratings_df.shape[1])
    ).toarray()

    # Calculate RMSE and MAE
    rmse, mae = calculate_rmse_mae(filtered_test_matrix, predicted_ratings_df.to_numpy())
    print(f"{cluster_name} - RMSE: {rmse:.4f}, MAE: {mae:.4f}")
```

Fig 4.5 The setup code (including RMSE and MAE formula) for the third sampling

In the third sample, clusters were segmented based on movie popularity into three groups: the top 10% of movies by rating count (Popular Movies), the next 40% (Moderate Movies), and the bottom 50% (Niche Movies). The dataset size remained consistent, with 25,000 top user ratings and 5,000 movies.

```
Popular Movies - RMSE: 1.9627, MAE: 1.7192
Moderate Movies - RMSE: 2.4581, MAE: 2.2342
Niche Movies - RMSE: 2.8649, MAE: 2.6670
```

Fig 4.6 RMSE and MAE for the 3rd sampling

The results mirrored the trends observed in the first cluster, with the top 10% (Popular Movies) achieving the best RMSE and MAE scores, followed by the Moderate group, and the bottom 50% (Niche Movies) performing the worst. This pattern underscores the model's ability to predict ratings more accurately for widely rated movies while struggling with sparsely rated items.

B. Test Discussion

These findings demonstrate that while SVD can uncover underlying patterns in dense datasets, its performance diminishes with sparse data and low-activity users. The high RMSE and MAE scores indicate that SVD alone is insufficient for creating a robust recommendation system, especially in real-world scenarios with diverse and incomplete data. The cold start problem further contributes to the model's limitations, as it cannot make accurate predictions for new users or items with little to no prior data.

To address these issues, hybrid recommendation models that integrate collaborative filtering with content-based approaches could provide a solution by leveraging item metadata and implicit feedback. Additionally, applying regularization techniques during matrix factorization can improve robustness against sparse data, while advanced models like Neural Collaborative Filtering (NCF) may better capture complex user-item relationships. Incorporating implicit interactions such as clicks or views and increasing the diversity of the dataset can also enhance the model's performance.

V. Conclusion

From the research conducted in this paper, it can be concluded that Singular Value Decomposition (SVD) is a fundamental approach for building recommendation systems, particularly in collaborative filtering. However, the testing results suggest that SVD alone, without the integration of additional methods, is insufficient for constructing a fully effective recommendation system. Challenges such as data sparsity and the cold start problem for new users were prominently highlighted during the evaluation, impacting the system's predictive accuracy and overall performance.

To address these challenges, future work should explore hybrid models that combine collaborative filtering with content-based approaches to mitigate sparsity and cold start issues. Additionally, incorporating implicit feedback, such as user interactions or viewing history, could enrich the dataset and improve recommendation quality. Advanced techniques, such as neural collaborative filtering or context-aware recommendations, may further enhance system performance by capturing complex, non-linear user-item relationships. These improvements can build upon the foundation established by SVD, paving the way for more robust and personalized recommendation systems.

VI. Appendix

For those who are interested in further development of this project, you can access the repository here.

VII. Acknowledgment

The author would like to express the deepest gratitude to the Lord Almighty for His guidance, wisdom, and blessings throughout the development period of this paper. It is through His kindness that mental barriers were broken down, clarity was achieved in solving complex problems, and the work was successfully completed. His presence has been a constant source of strength and inspiration, enabling perseverance during the most demanding phases of this endeavor. For this, the author is profoundly thankful.

The author would also like to extend heartfelt gratitude to all those who contributed to the preparation of this paper, including:

1. Ir. Rila Mandala, M.Eng., Ph.D., lecturer of the K1 IF2123 Linear Algebra and Geometry course, for his invaluable guidance and the knowledge imparted during the lectures,
2. The author's parents, for their continuous support, encouragement, and motivation throughout this process, and
3. The author's friends and peers, for their insightful ideas and encouragement during the making of this paper.

Their contributions and support have been vital in the successful completion of this work.

References

[1] IBM. *What is a recommendation engine?* IBM, https://www.ibm.com/think/topics/recommendation-engine. Accessed 28 December 2024.

[2] Roy, Deepjyoti; Dutta, Mala (2022). *A systematic review and research perspective on recommender systems*, Journal of Big Data.

[3] Ricci, Francesco; Rokach, Lior; Shapira, Bracha. *Recommender Systems Handbook* (3 ed.), Springer.

[4] Munir, Rinaldi. *Singular Value Decomposition (Bag. 1), Bahan Kuliah IF 2123*. Rinaldi Munir, https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2023-2024/Algeo-21-Singular-value-decomposition-Bagian1-2023.pdf. Accessed 29 December 2024

[5] Zonoozi, Ali. *What is Matrix Factorization?* BuiltIn, https://builtin.com/articles/matrix-factorization. Accessed 29 December 2024.

STATEMENT OF ORIGINALITY

I hereby declare that this paper I have written is my own work, not an adaptation or translation of someone else's paper, and not plagiarism.

Bandung, 27th December 2024

Haegen Quinston
13523109