Hybrid Music Recommender using LightGCN

S.K. Tuladhar and M. Dailey

Abstract— The music industry's rapid growth and the audience's increased access to music have made music recommender systems increasingly relevant. Various approaches, including content-based and collaborative filtering, have been explored for this. Graph Convolution networks (GCNs) perform well in recommendation using bipartite graphs, but struggle with cold-start problems. This paper explores the use of audio features from songs as content in GCNs, resulting in a hybrid method which improved recall at K values for music recommendation. The LightGCN, a state-of-the-art model with one layer of trainable embeddings, was used as the primary recommender. The musicnn library was used to extract audio features from songs which were used as the initialization values for the LightGCN embeddings for songs. The Spotify Million Playlist Dataset was used to train the model for link prediction. The model predicted which playlist a given song should belong to. This method showed a 5% increase in recall at K for both trained and fixed embedding settings for songs, indicating that audio embedding initializations improve the performance of the LightGCN. The recall at K for random initialization with trained embeddings was 0.51 and with fixed embeddings was 0.48. Both values increased to 0.57 upon using initializations extracted from audio. The hybrid method showed reasonable results for playlist continuation, the random initialization gave diverse results and extracted initialization gave consistent results based on similarity scores. However, it gave inconclusive results for item cold-start problems because no clear metric was identified for its evaluation.

Index Terms – **musicnn, item cold-start, audio embeddings, Spotify Million Playlist**

I. Introduction

RECOMMENDER systems are crucial for consumers to filter vast amounts of music content on the internet. With the rise of independent artists and the ease of access to all artists worldwide, a fast and effective way to recommend songs is needed. Recommender systems typically use collaborative filtering (CF) based on usage patterns and ratings.

Music recommendation has always been an extensive problem, and up until 2013, with the approach by den Oord, Dieleman and Schrauwen (2013) [1], Content-based (CB) recommenders were considered inferior to CF. Deep learning-based Graph Neural Networks (GNNs) like NeuMF, Neural Graph Collaborative Filtering (NGCF), and LightGCN perform well, but they struggle with the cold-start problem, which is of two types - item cold start and user cold start, depending on which information is missing [2]. There have been many hybrid models such as that of [3] which have combined CF with content data about items. However, until now, GNNs have mainly only used the CF approach.

This work was done as Sunny K. Tuladhar's Master's Thesis at Asian Institute of Technology (AIT), Pathum Thani 12120, Thailand, under the supervision of Matthew Dailey. The work was supported in part by Leapfrog Technologies, Charkhal, Kathmandu.

Sunny K. Tuladhar was with Asian Institute of Technologies, Pathum Thani 12120, Thailand. He is now with Leapfrog Technologies, Charkhal Rd, Kathmandu 44605.
(Email: sunnyktuladhar@gmail.com).

Matthew Dailey was also with Asian Institute of Technologies, Pathum Thani 12120, Thailand. He is now with Dexcom, USA. (Email: dailey.matthew@gmail.com).

In this paper, we took the LightGCN model [4] and used musicnn [5] to incorporate audio content features into node embeddings of songs, enabling a better hybrid recommendation model. The proposed architecture is shown in Fig. 1. Using the features extracted from musicnn we improved the model's performance and tackled the item-cold-start problem by giving the model data about the song content.

In order to achieve this, we followed the following steps:

i. Created a version of LightGCN incorporating audio features as embedding initializations.
ii. Used the data from the Spotify Million Playlist Dataset (MPD) along with the audio from Spotify's preview URL to download 30-second audio files for the experiments.
iii. Extracted the audio embeddings from the preview audio files using musicnn to initialize GNN embeddings of the songs and train the model.
iv. Evaluated the performance of the model in playlist continuation and item cold-start scenarios based on subjective analysis of the results.
v. Evaluated the experiment using recall@k metric with the LightGCN and compared the difference that embedding initializations make on the performance.

II. Literature Review

In this section, the two elements of the method, the recommender system and the audio feature embeddings which is used to initialize the node embeddings in our graph, are discussed. Among deep learning methods, GNNs have been shown to perform well for the task of recommendations. Audio embeddings generated by various methods also have been shown to capture the features to aptly represent the

ISSN: 2961-5992

content of the sound. Hence, combining these two elements was the prime motive of this paper.

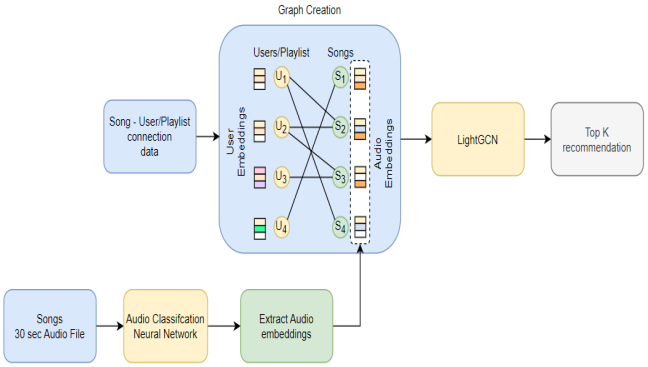

Fig. 1. Proposed method to introduce Audio Embeddings into LightGCN.

A. Recommender Systems

Recommender systems deduce users' preferences from user-item interactions or fixed attributes, and then suggest other products that users might be interested in. Formally, the main task of the recommender is to estimate or predict the user's preference for any item $i \in I$ represented by

$$y_{(u,i)} = f\left(h_*^u, h_*^i\right), \qquad (1)$$

where h_*^i is item i's learned representation, h_*^u is user u's learned representation, score function $f(\cdot)$ can implement any of a variety of operations including dot product, cosine and multi-layer perceptrons and $y_{(u,i)}$ is the preference or similarity score for user u on item i, which ideally represents the probability the two entities will interact. In this case, the item is the song, and the user is the playlist.

Recommender systems gained widespread use after the introduction of the matrix factorization (MF) collaborative filtering (CF) algorithm during the Netflix Prize competition. The inclusion of various kinds of supplementary information made this technique preferable to traditional nearest-neighbor algorithms for product recommendations [6]. Later, [1] used CF combined with latent audio features extracted from a CNN. [7] used neural networks to model the latent features in MF. [8] came up with NGCF, which make use of the user-item graph structure by generating embeddings on it.[4] removed unnecessary features from NGCF and improved its performance while making it less complex and created the LightGCN. This is the model we use for the experiments.

There are three types of recommender systems, depending on the type of information they use to make the recommendation. The categories are content-based (CB), collaborative filtering (CF), and hybrid.

CB recommendations are primarily based on comparisons between objects and auxiliary data provided by users [9]. CB filtering recommends items that have high similarity to past items the user has interacted with. This system is very efficient in cold start scenarios [10] but they are unable to provide personalized recommendations and are also limited in variety because there is insufficient data available about the user's profile.

CF is the most used approach in recommender systems. This method works on the principle that people with similar tastes will be interested in similar content. These systems base their recommendations on the preferences of like-minded users instead of the features of each item.

In hybrid recommender systems, multiple approaches are combined, and the benefits of each approach are used to offset the drawbacks of the others [10]. The method described in this paper can be considered a hybrid method as we incorporate content into the GCN in the form of audio embeddings.

B. Graph Neural Network-based Recommender Systems

One of the approaches that have led to rapid progress in recommender systems is the GNNs, which use a graph representation of the recommender data. User-item interactions can be aptly represented in the form of a bipartite graph between user and item nodes as shown in Fig. 2.

LightGCN is a Graph Convolutional Network built upon the NGCF, the previous state-of-the-art GCN-based recommender model [8]. Feature transformation and non-linear activation were removed from the NGCF model, resulting in LightGCN, which obtained better results. The basic GCN learns representations for nodes by smoothing features over the graph through iterative graph convolution. GCN aggregates features of neighbors to obtain a new representation of each target node:

$$\boldsymbol{e}_u^{(k+1)} = \mathrm{AGG}\left(\boldsymbol{e}_u^{(k)}, \{\boldsymbol{e}_i^{(k)} : \mathrm{i} \in \mathcal{N}_u\}\right), \qquad (2)$$

where, $\boldsymbol{e}_u^{(k)}$ and $\boldsymbol{e}_i^{(k)}$ are the refined embeddings of user u and item i, respectively, after k layers of propagation. $\mathcal{N}_u$ denotes the set of items that user u has interacted with. AGG is an aggregation function that considers the *k-th* layer's representation of the target node [4].

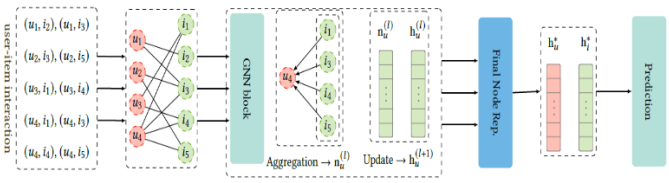

Fig. 2. GNN framework for user-item collaborative filtering. Reprinted from [11]

LightGCN uses a simpler weighted sum aggregator as shown in Fig. 3, and it abandons feature transformation and non-linear activations which have been shown to be burdensome for CF. In LightGCN, the graph convolution operation is defined as,

$$e_u^{(k+1)} = \sum_{i \in N_u} \left(\frac{1}{\left(\sqrt{(|N_i|)}\sqrt{(|N_u|)}\right)} \right) * e_i^{(k)},$$

$$e_i^{(k+1)} = \sum_{u \in N_i} \left(\frac{1}{\left(\sqrt{(|N_i|)}\sqrt{(|N_u|)}\right)} \right) * e_u^{(k)}, \qquad (3)$$

where N_u denotes the set of items user u has interacted with and N_i is the set of users that have interacted with item i [4]. The term $\frac{1}{\left(\sqrt{(|N_i|)}\sqrt{(|N_u|)}\right)}$ is the symmetric normalization

ISSN: 2961-5992

term that follows the standard GCN design of [12], which helps prevent the scale of embeddings from increasing with graph convolution operations.

All the layers are then combined as shown in Equation (4). α_k is set to $1/1+K$ as it gives good performance of the model [4].

$$\boldsymbol{e}_u = \sum_{k=0}^{K} \alpha_k \boldsymbol{e}_u^{(k)}; \quad \boldsymbol{e}_i = \sum_{k=0}^{K} \alpha_k \boldsymbol{e}_i^{(k)} \tag{4}$$

The final model prediction is the inner product of user and item final representations. It is used to rank items or users for recommendation generation. The equation is,

$$\hat{y}_{ui} = \boldsymbol{e}_u^T \boldsymbol{e}_i \tag{5}$$

LGCN uses the Bayesian personalized ranking loss [13]

$$L_{BPR} = -\sum_{u=1}^{M} \sum_{i \in \mathcal{N}_u} \sum_{j \notin \mathcal{N}_u} \ln \sigma\left(\hat{y}_{ui} - \hat{y}_{uj}\right) + \lambda ||E^{(0)}||^2 \tag{6}$$

where λ controls the L2 regularization strength.

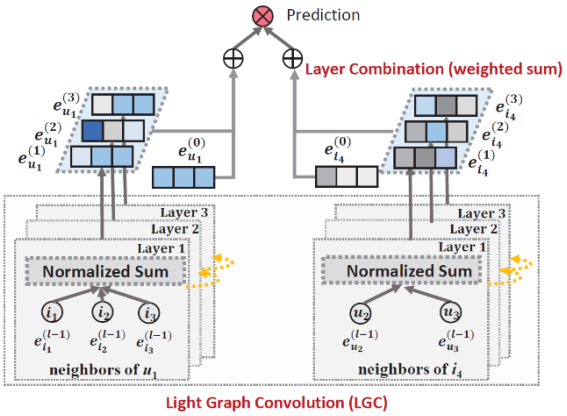

Fig. 3. LightGCN architecture. Reprinted from [4].

C. Audio Embeddings

For this paper, the audio embeddings were extracted from the songs using the Python library *musicnn* [5] which contains models pre-trained on the Million Song Dataset (MSD) [14] and the MagnaTagATune (MTT) [15] dataset. The model can be used for out-of-the-box music audio tagging. This was used to extract embeddings from audio which was our goal.

The basic building blocks of the *musicnn* model can be seen in Fig. 4. The model uses a log-mel spectrogram, which is fed into a music-motivated CNN front-end to capture the timbral and temporal features of the audio. These features are fed to the dense layer middle portion of the model, which extracts higher-level representations from the previous layers. The final temporal pooling back-end predicts tags from the extracted features. The penultimate layer is derived from this back-end. These are the features used in this experiment to initialize the LightGCN embeddings.

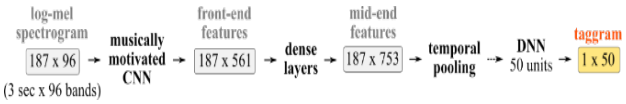

Fig. 4. Overall building blocks of musicnn model. Reprinted from [5]

III. Methodology and Experimental Design

For our experiment, content information was injected in the form of audio feature embeddings extracted from musicnn into our LightGCN model initialization for songs as shown in Fig. 1. In this section, the entire process, from data acquisition, pre-processing, audio embedding extraction, recommendation prediction, and evaluation are discussed.

A. Dataset

The dataset used is the Spotify MPD [16], which contains 1,000,000 playlists. A playlist includes playlist title and the titles of the songs. The dataset has over two million unique songs and over 66 million edges. The edges are links between songs and playlists. The playlists were created by users on the Spotify platform between January 2010 and October 2017. We used this information later to simulate the cold-start problem taking songs released after 2018 that are not present in the dataset.

Since the original dataset is extensive, we used a smaller subset of different sizes as created by [17] for our experiments. The data subset was created by calculating its K-core which is "The largest connected sub-graph of a graph with every node having a degree of at least K" [17]. Here we used K-core 10 and removed songs with no audio previews to get the required dataset. Our Spotify MPD K-core 10 dataset has 71,233 playlists and 27,687 songs after removing songs with no audio previews. It has 1,812,878 undirected edges between songs and playlists.

B. Preprocessing

In this section, we discuss the process for converting of the raw JSON files, which is the default format of the MPD, specifying playlists and songs as nodes into to a PyTorch Geometric Graph, along with the audio embedding extraction process.

Graph creation

The graph representation of the dataset was created by converting the JSON file of the Spotify MPD into a PyTorch Geometric file with the help of Stanford-SNAP library [18]. The graph is bipartite, so there are no connections between similar nodes i.e., no connection between songs or between playlists themselves.

Audio embeddings

The audio embeddings were extracted from the songs with Spotify's available 30-second previews using the Spotipy library for Python [19]. The songs that did not have audio previews were discarded.

The embeddings of the audio were extracted using *musicnn*. It had two pre-trained weights MTT-musicnn and MSD-musicnn, which were obtained by training the MTT

ISSN: 2961-5992

and MSD, respectively. *Musicnn* generates song-level tags from audio. The tags include the instrument, style, vocal types, and many more based on the content of the audio file. For this case, we extracted the 200-unit embeddings from the penultimate layer of *musicnn*. These 200 elements are reduced to 64 using Principal Component Analysis (PCA), as our LightGCN model is designed for 64-element embedding initialization.

Combining Audio Embeddings and LightGCN

Our main objective is to feed content information into LightGCN. We do this by initializing song node embeddings in the GCN using the audio embeddings of the songs extracted by *musicnn*. The task we performed was link prediction. We predicted links between two nodes (a song and a playlist) based on their similarity scores. We used the dot products between embeddings of the two nodes (song and playlist) from the final layer of the model to represent how likely the song is to end up in that playlist. Our equation for the similarity is the dot product between song and playlist embeddings as shown below,

$$sim(p, s) = x_p^K \cdot x_s^K \quad (7)$$

where p is a playlist, s is a song, x_p^K is the embedding for the playlist, and x_s^K is the embedding for the song, both of K dimensions.

GCNs create node embeddings for each playlist and each song. We initialized song embeddings as the audio embeddings generated from *musicnn*. We initialized playlist embeddings using a random normal distribution. Then, we used two methods to train the model. First, *fixed embeddings*, where we froze the song embeddings by giving them a gradient of 0 during training. Then we trained the model, and it learned only the playlist embeddings. The song embeddings remained unchanged throughout the training. Second is *trained embeddings*, where we trained both song and playlist embeddings. Here, the model learned new embeddings for both songs and playlists while it trained. Loss function used in both cases was the BPR loss.

C. Evaluation method

The model was evaluated using two methods. First was calculating the recall@k. The second was the subjective analysis of the results for two tasks, playlist continuation and song cold-start problem.

Recall

To assess the model performance, we used recall@k. Recall@k is the ratio of the number of songs in the top K recommendations that are relevant to the total number of songs that are relevant according to the ground truth.

$$\text{Recall@k} = \frac{Songs\ in\ top\ k\ that\ are\ relevant}{Total\ relevant\ songs}, \quad (8)$$

For a specific playlist p it can be expressed as

$$Recall@k \text{ for playlist } p = \frac{\left|P_p \cap R_p\right|}{\left|P_p\right|}, \quad (9)$$

where, P_p is the set of positive items (songs) the playlist includes and R_p is the set of songs recommended by the model. For the top K recommendations $|R_p| = K$. Songs that are already in the playlist (training set) were excluded. The final recall@k value was calculated by averaging recall across all playlists.

Subjective Evaluation

Since we are dealing with music, it is difficult to rely only on the recall values, as there are subjective nuances to music and its recommendation. To address this, we performed two types of subjective evaluation. The first recommended songs for a particular playlist (playlist continuation) and the second predicted the playlists for a particular song (item cold-start).

For playlist continuation, we evaluated the recommendation of songs given for a custom playlist-of-interest. We obtained the embeddings of this playlist. Then we obtained the top K songs, not present in the playlist, that have the highest similarity score with the playlist by calculating the dot product of the embedding of the playlist with all the song embeddings in the dataset, as shown in Equation 10. Recommended songs were then subjectively analyzed based on genre given the songs that it was trained on. Specifically,

$$sim(poi, s) = x_{poi}^{64} \cdot x_s^{64}\ ;\ poi \in P\ ,\ s \in S, \quad (10)$$

where poi is the playlist of interest also present in the playlist dataset P , s is a song in the dataset S, x_{poi}^{64} is the 64-dimension embedding for the playlist and x_s^{64} is the 64-dimension embedding for the song.

For the item cold-start, we evaluated the playlists recommended for a "cold" song. Given a song with no previous links, the model predicts the best playlist it should belong to. We evaluated the cold-start performance through this. Since the train-test validation split used is random, and each of the splits includes all the nodes. The method was introducing new songs from after 2018, which were not present in our dataset and subjectively analyze those songs' playlist prediction. Embeddings of the songs were generated using the *musicnn* library. The similarity scores were checked against each playlist's embeddings as shown below. The model predicted the top playlists the song could belong to, based on the top similarity scores, and the results were subjectively analyzed. Specifically,

$$sim(p, soi) = x_p^{64} \cdot x_{soi}^{64}\ ;\ p \in P\ ,\ soi \notin S, \quad (11)$$

where soi is the cold song of interest, which is not in the song dataset S, p are the playlists in the playlist dataset P, x_p^{64} is the 64-dimension embedding for the playlist and x_{soi}^{64} is the 64-dimension embedding for the selected song.

ISSN: 2961-5992

IV. RESULTS

In this section, we discuss the experiments performed regarding the LightGCN-based recommendation system initialized with various song node embedding initializations. For the experiments, LightGCN was run with three different initializations, random normal (default), MTT-musicnn, and MSD-musicnn.

LightGCN was trained on the two datasets with random normal and the two musicnn embeddings initializations for songs with fixed and trainable embedding for the extracted featured song initializations and the results were compared. The aggregation function for the GCN is addition and the optimizer used is Adam. For negative sampling for loss calculation, we used random sampling to save computation costs, as there were over 27,000 songs. All experiments were run with an embedding size of 64, and all training was done at a learning rate of 0.001. The number of LightGCN layers (k) was set to 3, as recommended. All these hyperparameters were selected based on the implementation by [17].

TABLE I
RECALL@500 VALUE FOR K-CORE 10 DATASET FOR TWO INITIALIZATIONS IN TRAINED AND FIXED SETTINGS

Song embedding initialization	Trained	Fixed
Random Normal	0.51	0.48
Musicnn-MTT	0.56	0.53
Musicnn-MSD	***0.57***	***0.57***

A. Experiments on K-core 10 Graph dataset

We trained the K-core 10 dataset for 100 epochs with 3 layers, a batch size of 1024, and an embedding dimensionality of 64 in all the different embeddings and trained/fixed settings. We calculated recall@k at K = 500, which amounts to 2% of the total songs. K was selected as such because the total number of songs was very high. We split the dataset into 70% training, 15% validation, and 15% test. Each split has all the nodes so only the edges were split.

We trained it first with embeddings of both songs and playlists initialized to random normal values. We obtained a recall@500 with random normal initialization of 0.51. With the random normal song embeddings fixed, we obtained a recall@500 value of 0.48.

Then we trained the model with song embeddings initialized to the two different *musicnn* embeddings (MTT and MSD) while initializing playlists with the default random normal distribution. For both embeddings, we trained the model in two settings. The first with song embeddings fixed, and the second where song embeddings were allowed to train. The fixed MTT-musicnn song initializations gave a recall@500 of 0.53, while trainable MTT-musicnn improved it to 0.56. However, using fixed and trainable MSD-musicnn embeddings further improved both recall to 0.57. The results are shown in TABLE I, which was the main finding of this paper. This showed that introducing content embeddings increased the model performance.

The trained embeddings reached a high value of validation recall very early and then started to overfit early compared to the fixed versions, as can be seen in Fig. 5. We see that both of *musicnn* embedding song initializations always performed better than the random normal initializations. The MSD embeddings performed better than the MTT embeddings which could be because MSD has better accuracy in the task of audio-tagging. This could imply better audio-tagging embedding models could result in better performance.

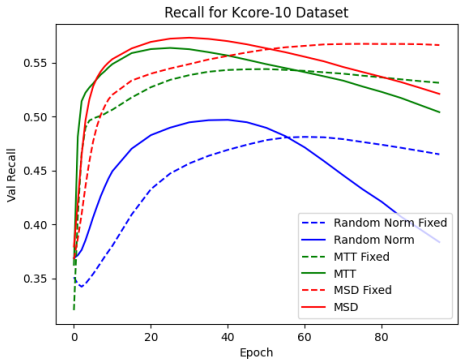

Fig. 5. Validation recall values at different song embedding initializations at fixed and trainable song embeddings.

B. Subjective Experiments

We conducted two experiments to subjectively analyze the model's performance. One recommends songs based on a given playlist (Playlist continuation), and the second recommends playlists where a new unlinked song will most likely belong to (Item cold-start). All these experiments were conducted on the K-core 10 dataset and the best-performing model, which was MSD.

Songs for playlists (Playlist continuation)

To analyze the playlist continuation, we evaluated the model's recommendations of songs for a given playlist. We took a user's personal playlist with 8 songs which were present in MPD. The playlist was added to the graph, and the model was trained with MSD trained embeddings. The 64-dimensional playlist embeddings were obtained for the playlist. The dot product of the playlist was calculated against all the songs in the graph to obtain similarity scores. The original playlist is shown in TABLE II.

TABLE II
ORIGINAL PLAYLIST "MATT" FOR PLAYLIST CONTINUATION

Songs in the original playlist	Genre
The Man Who Sold the World, David Bowie	Rock
I Wanna Be Your Dog, The Stooges	Rock
Sweet Jane, The Velvet Underground	Rock
Paranoid, Black Sabbath	Rock
Killing in the Name, Rage Against the Machine	Rap Metal
Currency, The Black Angels	Alternative
Cold Cold Cold, Cage the Elephant	Alternative

ISSN: 2961-5992

TABLE III
PLAYLIST CONTINUATION RECOMMENDATIONS BASED ON MSD-MUSICNN AND RANDOM EMBEDDINGS FOR PLAYLIST "MATT".

MSD Initialized Recommended Songs (Similarity Score σ = 0.68)		
Song	**Genre**	**Similarity Score**
Lady Marmalade, Patti LaBelle	Funk	12.73
Break On Through (To The Other Side), The Doors	Rock	11.93
Burnin' for You, Blue Öyster Cult	Classic Rock	11.63
Highway to Hell, AC/DC	Hard Rock	11.43
Long Cool Woman (In A Black Dress), The Hollies	Swamp Rock	11.27
Faithfully, Journey	Classic Rock	10.69
Suite: Judy Blue Eyes, Crosby, Stills and Nash	Rock	10.65
Ramble On, Led Zeppelin	Rock	10.65
Random Normal Initialized Recommended Songs (Similarity Score σ = 2.31)		
Song	**Genre**	**Similarity Score**
Break On Through (To The Other Side), The Doors	Rock	16.73
Lady Marmalade, Patti LaBelle	Funk	14.42
Highway to Hell, AC/DC	Hard Rock	13.83
Don't Lean On Me, The Amity Affliction	Metal	12.43
Ramble On, Led Zeppelin	Rock	12.27
Take Me Home Tonight, Eddie Money	Hard Rock	10.19
A Horse with No Name, America	Folk Rock	10.05
We Didn't Start the Fire, Billy Joel	Pop Rock	9.65

The original playlist was mainly a rock playlist with all the songs in the "Rock," "Metal" or "Alternative" genres. In the results from the MSD initialized recommender system, shown in TABLE III, we got almost all songs from the rock genre and its variations such as "Classic Rock," "Folk Rock," "Hard Rock," and "Swamp Rock". The similarity scores were less varying **(**standard deviation**,** σ = 0.68) but consequently the recommended songs were less diverse.

The random normal initialized recommender also had songs that are mostly "Rock" and "Metal" shown in Table III. However, it also recommended one Funk song. The similarity scores were more diverse **(**σ = 2.31) and so were the recommended songs. These were arguably better recommendations in terms of variation.

Playlist for songs (Item cold-start)

This task was to find the best playlists that a new song would belong to. This experiment was conducted using the fixed MSD-musicnn embeddings. We chose fixed because when we introduce a new song with no links its embeddings cannot be trained. The problem of introducing a song with no links is our item cold-start problem. To analyze the cold-start performance of the model we introduced songs that were not present in the dataset, songs released after 2018, shown in TABLE IV. We then extracted the 200-dimension embeddings of these new songs using the MSD-musicnn. These embeddings were reduced to 64 dimensions using PCA by fitting them with the respective 200-dimension embeddings of the originally extracted embeddings of the songs in the K-core 10 dataset.

Then we calculated the similarity scores of these song embeddings against all the playlist embeddings which were generated after being trained on MPD. We calculated the similarity scores by taking the dot product between the embeddings of the songs and the playlists. Then we selected the top five playlists these songs would belong to, based on the highest similarity scores.

ISSN: 2961-5992

TABLE IV
SONGS SELECTED FOR THE SONG COLD-START EXPERIMENT.

Song Artist	***Song Name***	***musicnntags***	***Genre (Google)***
Singto Numchok	*I just wanna pen-fan you daibor*	*guitar, male, male vocal*	*Pop*
Stone Temple Pilots	Three Wishes	guitar, male, pop	Rock/Folk
Dua Lipa	Levitating	techno, pop, loud	Pop/Electro
Adele	Easy On Me	guitar, piano, vocal	Pop
Metallica	Lux Æterna	fast, techno, rock	Speed Metal
Drake ft. Lil Durk	Laugh Now Cry Later	techno, electronic, male	Hip-Hop/Rap
Taylor Swift	Anti-Hero	female, woman, female vocal	Pop/Rock

TABLE V
COLD-START PLAYLIST RECOMMENDATIONS BASED ON MSD EMBEDDINGS.

"Cold" Song Artist	**Song Name**	**Playlist 1**	**Playlist 2**	**Playlist 3**	**Playlist 4**
Singto	I just wanna pen-fan you daibor	miami	Harry Potter	pop	lit
Stone T	Three Wishes	Acoustic Mix	2014	slow	June
Dua L.	Levitating	boy bands	classics	Pop songs	boy band
Adele	Easy On Me	fempower	Classique	calm	sad
Metallica	**Lux Æterna**	**Heavy Metal**	**Rock Music**	**Dante**	**metal**
Drake	Laugh Now Cry Later	Rapman	Beyoncé	Beyoncé	Tunes
Taylor S.	Anti-Hero	Texas Country	new country	Country	Texas forever

All the audio data for the new songs were taken from YouTube in the form of MP3 files. The songs were chosen to incorporate the most common genres such as rock, hard rock, pop, and hip-hop. All the songs are among the most popular songs in the genre and composed by popular artists, so they could be subjectively judged by general users, who have a high chance of having listened to it. All songs were released after 2018, so they do not have any existing link to the playlists in the training dataset. The embeddings were extracted using MSD-musicnn from the full length of the song and not just the 30-second audio previews as in the training data.

The recommended playlists are shown in TABLE V. The model correctly put Stone Temple pilots "Three wishes" in acoustic Mix as it is an acoustic song. It distinctly recognized the Metallica's "Lux Æterna" as Hard Rock and Metal. Songs such as Adele's "Easy on Me"' also seemed to be placed in appropriate playlists called "calm"' and "sad." Dua Lipa's song also seemed correctly placed even though the name is "boy bands" as this was a pop song. Despite this, the pop song playlists were not as good, as the playlists recommended were too diverse and seemed random. Further research will be required for any conclusions.

C. Discussion

In this section, we discuss the results obtained in the above experiments and analyze the performance of the model. We analyzed the clustering capabilities of our model and its objective evaluation using recall@k. Its subjective evaluation has already been discussed.

Clustering

From Fig. 6, we can see that all three embedding initializations did a fine job of creating suitable embeddings for the playlists. The 64 dimensions of the playlist embeddings were taken from the best-performing epoch after the model was trained with the different initializations for songs. These 64 dimensions were reduced to two dimensions using t-SNE [20]. These clusters were labeled using the name of the playlists. If a playlist contained the words "Rock," "Pop", "Country", "Rnb" or "Rap" it was assumed to have songs from that genre. No genre or text information was explicitly given to the model. Since there was no direct connection between the two playlists as the graph is bipartite, this clustering occurred completely based on the song placement in the playlists.

ISSN: 2961-5992

While the *musicnn* initializations could provide the model with genre information, as they are the audio feature embeddings, it was surprising to see that the model learned similar clustering even with random normal initializations. Based on the well-clustered graph for genre, even with random normal initializations, LightGCN embeddings seemed to perform well in generating embeddings for playlists.

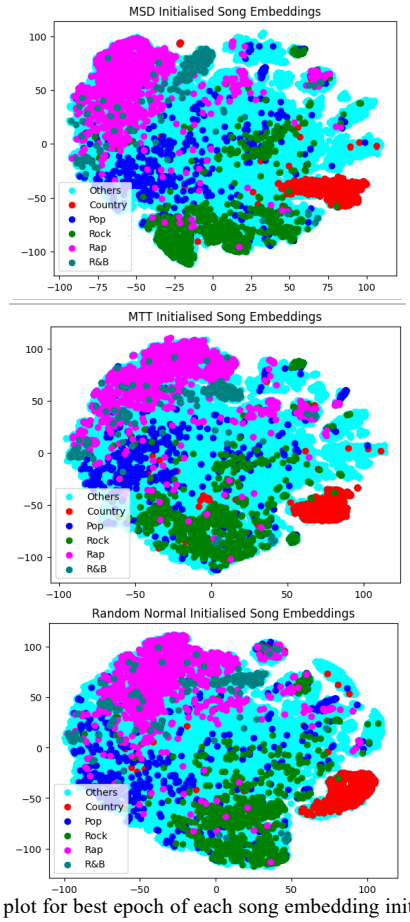

Fig. 6. t-SNE plot for best epoch of each song embedding initializations for the playlist data. Playlists grouped by genre.

Recall

Our main task was to predict links between the songs and playlists as measured by recall@k. For this, we saw a significantly higher performance for the *musicnn* song embedding initializations compared to random normal initializations. When keeping the song embeddings fixed, while using both the extracted and random normal embeddings, we saw the differences in the performance, as shown in Table I. Based on the given performance differences between the initializations with music embeddings, there seemed to be a significantly better performance with content information introduced in the dataset. The *musicnn* embedding initializations always performed better than the random normal ones.

The performance of extracted song embeddings was analyzed with and without further training. Since the performance difference was not very high with fixed and trained song embeddings for the same embedding type, it shows that the *musicnn* initialization values were more representative of songs and gave good results even when they were not updated.

V. CONCLUSION

In this paper, we introduced GNN methods and their impressive performance in the task of recommendation. GNNs have not been well-researched in terms of CB recommendations. For this, we introduced audio content features as embeddings initializations from *musicnn* into LightGCN. A comparison of the performance with and without the extracted embedding initializations showed that the extracted embedding initialization methods performed better for general recommendation and playlist continuation. Item-cold-start for songs still require further research.

A. Conclusion

We saw that the extracted embeddings always perform better than the random normal initialization values in the music recommendation experiments. Thus, introducing content in the form of extracted embeddings improves recommendation performance, as measured by the recall@k metric.

Upon subjectively analyzing the recommended songs that could belong in a playlist for a specific case, we saw that both embedding initializations (random and MSD-musicnn) gave reasonably good recommendations. This was based on the genre of the songs in the original playlists and the genre of the songs that were recommended. Song recommendation is a tricky matter, and solely relying on the genre or the similarity score would not be a good metric, as sometimes variation is a good thing for the listener.

However, in the task of item-cold start, the model did not produce convincing results for pop songs but did give reasonable results for rock and metal songs.

B. Limitations and Future Work

Some of the limitations of this paper are as follows.

i. The current experiments were performed only with 30-second audio clips of songs. A much more complete analysis might take into account the entire song audio.

ii. Varying amounts of embeddings could be tried out. Changing the embedding size to 32, 64, or 128 and evaluating the changes in performance could provide valuable insights into the effects of its embedding size.

iii. A more robust metric to measure the performance of a recommender system could be developed. The metric should be something more transparent than recall@k but easier to justify than the type of subjective analysis done here based mainly on song genre.

iv. Also, an end-to-end network combining *musicnn* and LightGCN in which we input audio files and generate embeddings within the network could also yield better results.

ISSN: 2961-5992

REFERENCES

[1] A. den Oord, S. Dieleman, and B. Schrauwen, 'Deep content-based music recommendation', *Adv Neural Inf Process Syst*, vol. 26, 2013.

[2] L. Bernardi, J. Kamps, J. Kiseleva, and M. J. I. Müller, 'The continuous cold start problem in e-commerce recommender systems', *arXiv preprint arXiv:1508.01177*, 2015.

[3] R. Kiran, P. Kumar, and B. Bhasker, 'DNNRec: A novel deep learning based hybrid recommender system', *Expert Syst Appl*, vol. 144, p. 113054, 2020.

[4] X. He, K. Deng, X. Wang, Y. Li, Y. Zhang, and M. Wang, 'Lightgcn: Simplifying and powering graph convolution network for recommendation', in *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*, 2020, pp. 639–648.

[5] J. Pons and X. Serra, 'musicnn: Pre-trained convolutional neural networks for music audio tagging', *arXiv preprint arXiv:1909.06654*, 2019.

[6] Y. Koren, R. Bell, and C. Volinsky, 'Matrix factorization techniques for recommender systems', *Computer (Long Beach Calif)*, vol. 42, no. 8, pp. 30–37, 2009.

[7] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua, 'Neural collaborative filtering', in *Proceedings of the 26th international conference on world wide web*, 2017, pp. 173–182.

[8] X. Wang, X. He, M. Wang, F. Feng, and T.-S. Chua, 'Neural graph collaborative filtering', in *Proceedings of the 42nd international ACM SIGIR conference on Research and development in Information Retrieval*, 2019, pp. 165–174.

[9] S. Zhang, L. Yao, A. Sun, and Y. Tay, 'Deep learning based recommender system: A survey and new perspectives', *ACM Computing Surveys (CSUR)*, vol. 52, no. 1, pp. 1–38, 2019.

[10] Z. Batmaz, A. Yurekli, A. Bilge, and C. Kaleli, 'A review on deep learning for recommender systems: challenges and remedies', *Artif Intell Rev*, vol. 52, no. 1, pp. 1–37, 2019.

[11] S. Wu, F. Sun, W. Zhang, X. Xie, and B. Cui, 'Graph neural networks in recommender systems: a survey', *ACM Computing Surveys (CSUR)*, 2020.

[12] T. N. Kipf and M. Welling, 'Semi-supervised classification with graph convolutional networks', *arXiv preprint arXiv:1609.02907*, 2016.

[13] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme, 'BPR: Bayesian personalized ranking from implicit feedback', *arXiv preprint arXiv:1205.2618*, 2012.

[14] T. Bertin-Mahieux, D. P. W. Ellis, B. Whitman, and P. Lamere, 'The million song dataset', 2011.

[15] E. Law, K. West, M. I. Mandel, M. Bay, and J. S. Downie, 'Evaluation of algorithms using games: The case of music tagging.', in *ISMIR*, 2009, pp. 387–392.

[16] C.-W. Chen, P. Lamere, M. Schedl, and H. Zamani, 'Recsys challenge 2018: Automatic music playlist continuation', in *Proceedings of the 12th ACM Conference on Recommender Systems*, 2018, pp. 527–528.

[17] B. Alexander, J.-P. Chou, and A. Bansal, 'Implement Your Own Music Recommender with Graph Neural Networks (LightGCN)', Dec. 2021. [Online]. Available: https://medium.com/@benalex/implement-your-own-music-recommender-with-graph-neural-networks-lightgcn-f59e3bf5f8f5

[18] J. Leskovec and R. Sosič, 'SNAP: A General-Purpose Network Analysis and Graph-Mining Library', *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 8, no. 1, p. 1, 2016.

[19] P. Lamere *et al.*, 'vssousa/spotipy: v1.0', Sep. 2017, *Zenodo*. doi: 10.5281/zenodo.886524.

[20] L. der Maaten and G. Hinton, 'Visualizing data using t-SNE.', *Journal of machine learning research*, vol. 9, no. 11, 2008.

ISSN: 2961-5992