
GNN-based Spotify Playlist Recommendation

Qianchi Huang
Carnegie Mellon University
Pittsburgh, PA 15213
qianchih@andrew.cmu.edu

Shubham Singh
Carnegie Mellon University
Pittsburgh, PA 15213
shubham4@andrew.cmu.edu

Abstract

This project tackles the problem of personalized Spotify playlist recommendations, aiming to improve the user experience by generating high-quality automatic playlist extensions. Traditional methods often fail to capture nuanced relationships between tracks and playlists, leading to less effective recommendations. Using the Spotify Million Playlist Dataset, we conceptualized playlists and tracks as nodes in a bipartite graph. Starting with LightGCN as the baseline model, we introduced Graph Attention Networks to weight connections dynamically and a Multi-Layer Perceptron (MLP) for non-linear feature refinement. This architecture aggregates information across nodes and computes embeddings for playlists and tracks, enabling effective similarity-based recommendations. Our approach showcases enhanced performance from the baseline model, with recall improvement and substantial reductions in training and validation loss. These results showcase the strength of GAT in capturing high-order dependencies and the power of MLP in refining embeddings. This work not only highlights the benefits of graph-based approaches, but also sets a foundation for extending similar techniques to other recommendation systems, such as movies or e-commerce. By enhancing playlist personalization and music discovery, our model holds practical value in increasing user satisfaction and platform engagement.

1 Introduction

Spotify has been a leading platform in the music streaming industry. It attracts users with its ability to deliver effective and personalized music experiences. One of its key attractions lies in its ability to create tailored playlists according to each individual's listening habits. As deep learning continues to gain popularity in the field of recommendations, it is now being used in a variety of applications, including music, due to its capability to analyze large-scale data and discover patterns and intricate relationships. Techniques like convolutional neural networks and recurrent neural networks have been successfully applied to enhance music recommendations by identifying patterns in audio features and user behavior. (Yin 2023) This project proposes generating recommended songs to the playlists from the input of songs within an existing playlist using deep learning models. By using public Spotify playlists as our primary data source, we aim to train and test with this dataset to create a recommendation engine that successfully delivers a context-based playlist recommendation.

2 Literature Review

The project task involves automatic playlist continuation. Some authors have approached the problem based on Markov chains to model the transitions between songs in playlists. (McFee et al. 2011) While these approaches have been shown to perform well in terms of log likelihood, recent research has found little evidence that the exact order of songs actually matters to users, while the ensemble of songs in a playlist and direct song-to-song transitions seems to matter. (Chen et al. 2018)

Schedl et al. have recently identified the task of automatic music playlist continuation as one of the grand challenges in music recommender systems research. (Schedl et al. 2018) Research on music recommendation systems using deep learning models has been extensive. Maheshwari has implemented recommendations using audio features based on network theory, subsequently using deep learning algorithms to enhance recommendation performance. (Maheshwari, 2023) Another research performed by Yin explored the use of CNNs alongside mel spectrograms to categorize music tracks and enhance music track recommendations, demonstrating the effectiveness of deep learning in capturing intricate audio patterns. (Yin 2023) However, both studies generated recommendations by approaching individual song features; they did not consider the collective perspective within playlists, which limits their effectiveness in automatic playlist continuation.

3 Baseline Selection

The baseline model selected for this project is the LGCN based model integrating collaborative filtering from 2023. Neural recommender models, especially those based on embeddings and deep neural networks, have shown superior performance compared to traditional collaborative filtering techniques. GNNs are designed to leverage the graph structure of data and can capture higher-order dependencies between nodes and edges as well as node- and edge-level features to learn embeddings or representations of nodes, edges, or (sub)graphs. In this case, each node carries a message and then propagates these messages along the edges of the graph, aggregating and updating many times. It can also be scaled effectively for larger datasets, making it suitable for practical applications like music recommendation. In the Spotify playlist dataset, the model uses embeddings to represent both user preferences and track features in a shared space, which allows it to learn nuanced relationships between them through deep learning techniques. This baseline model is a robust starting point that allows us to measure performance accurately against established metrics and determine how any modifications or enhancements affect recommendation quality.

3.1 Model Description

The baseline model described in the data is a graph-based recommender system that uses graph neural network (GNN) layers to capture relationships between nodes (playlists and tracks) in a bipartite graph. Specifically, it explores a special type of GNN layers: LightGCN (LGConv). Below is a mathematical overview of the component and how it is used in the baseline model.

1. Graph Structure and Node Embeddings Let $G = (V, E)$ represent the bipartite graph, where V consists of two types of nodes: playlists and tracks, and E represents edges connecting playlists and tracks. Each node $v \in V$ has an embedding $h_v \in \mathbb{R}^d$, where d is the embedding dimension. The model learns an embedding for each playlist and track by propagating messages between neighboring nodes.

2. LightGCN Layer (LGConv) Light Graph Convolutional Network (LightGCN) is an efficient variant of traditional GCNs, tailored for large-scale recommendation tasks. Unlike standard GCNs that rely on weight matrices and activation functions, LightGCN removes these elements to focus solely on embedding propagation, thus reducing complexity and preventing over-smoothing, where node embeddings become indistinguishable. In LightGCN, user and item nodes aggregate information from neighboring nodes over multiple layers, capturing both direct and indirect interactions. The final embedding for each node combines information from all layers, allowing multi-level insights with minimal computation.

$$h_v^{(k+1)} = \sum_{u \in \mathcal{N}(v)} \frac{h_u^{(k)}}{\sqrt{|\mathcal{N}(u)|} \cdot |\mathcal{N}(v)|} \quad (1)$$

The formula showing $h_v^{(k+1)}$ is the embedding of node v at layer $k+1$, and $\mathcal{N}(v)$ is the set of neighboring nodes of v . The embeddings are aggregated across layers to form the final representation.

$$e_i^{(k+1)} = \sum_{j \in \mathcal{N}(i)} \frac{1}{\sqrt{|\mathcal{N}(i)|} \cdot \sqrt{|\mathcal{N}(j)|}} e_j^{(k)} \quad (2)$$

3. Embedding Similarity for Recommendations After obtaining the node embeddings, similarity between playlists and tracks is computed using a dot product: where h_p and h_t are the embeddings for playlist p and track t , respectively. Higher scores indicate a higher likelihood that a track is relevant to a playlist.

$$\text{score}(p, t) = h_p^T h_t$$

4 Dataset

The Spotify Million Playlist Dataset initially released by Spotify Research solely for the RecSys Challenge 2018, the dataset was later made publicly available. The dataset consists of one million Spotify playlists, complete with track information, artist details, and album data. In total, the dataset has over 2 million unique songs from nearly 300,000 artists. The full dataset folder is roughly 34 gigabytes, made up of 1,000 JSON files containing 1,000 playlists each.

We can conceptualize this data as a graph, where each node represents a playlist, track, artist, or album, and edges link playlists with the tracks, artists, and albums they contain. Additionally, edges link tracks to their respective artist(s) and album, and artists to their albums. Being concerned with song recommendations for playlists, our goal is to learn to predict the playlist-track edges well.

We want to note that the initial set-up (with 4 node types), we will exclude the artist and album nodes. This leaves us with a bipartite graph between playlists and tracks, where an edge indicates that a track is contained in a playlist.

The resulting graph still has 512 thousand nodes and over 3.3 million edges. Unfortunately, given our computational resources on Colab (25 GB system RAM, 15 GB GPU RAM), this graph is too large to train our models on. As a result, we will need to identify a subgraph to continue our analysis. To do so, we use the graph theory concept of a “K-core”, which is defined as the maximally connected subgraph in which all nodes have degree of at least K. This means that any node that is not in the K-core must have a degree strictly less than K. Taking the K-core of our graph allows us to identify a central, highly-connected subgraph of the original graph, where each playlist contains at least K tracks, and each such track is a member of at least K-1 other playlists.

Thus, the subgraph reduces the size of our node-space by a factor of 15 and halves the number of edges. Moreover, this subgraph is significantly denser: the average node degree in the subgraph is approximately 90, almost seven times larger than in the original graph (approximately 13). This should significantly aid the learning process, allowing our methods to better capture and propagate information through the graph structure, resulting in better node embeddings, which are essential for accurate predictions.

Specifically, our final subgraph contains 34,810 nodes (22,563 playlists and 12,247 tracks) with 1,578,286 edges. We briefly recap the steps for obtaining this subgraph below:

1. Load the first 50 JSON files, containing 50,000 playlists.
2. Drop artist and album information to create a bipartite graph between playlists and tracks.
3. Compute the 30-core of this graph.

5 Methods and Approaches

The model begins with graph construction, where playlists and tracks are represented as nodes. Edges between nodes indicate associations, such as a track belonging to a playlist. The baseline model, LightGCN, propagates and aggregates features across the graph to generate embeddings, leveraging the graph’s structural relationships.

5.1 Model Improvements

To enhance the baseline model, we employed a Graph Attention Network (GAT). GAT introduces attention-based weighting of node connections, enabling the model to assign varying importance to

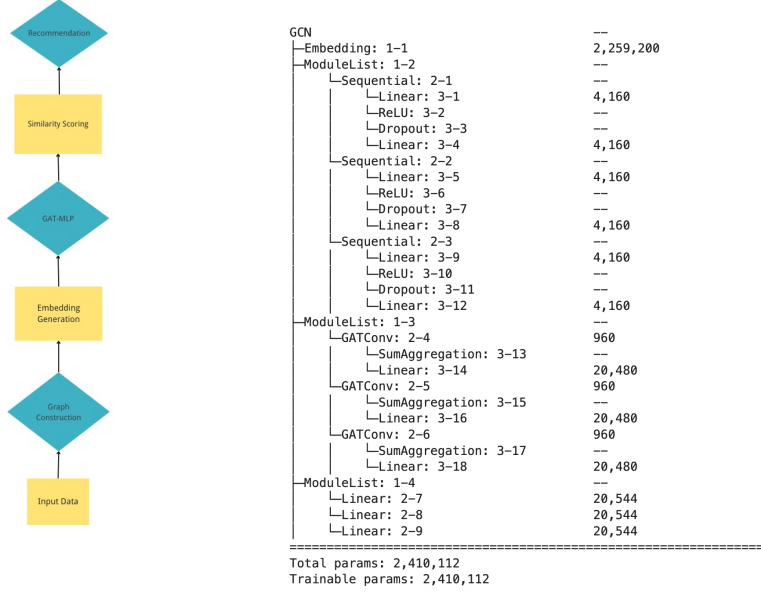
different neighbors. By applying multiple graph convolution layers, the model aggregates information from nodes further apart in the graph, capturing its global structure. Below is the embedding update at layer $k+1$ in GAT convolution.

$$\mathbf{e}_i^{(k+1)} = \Theta \mathbf{x}_i^{(k+1)} + \mathbf{B} \quad (3)$$

$$\mathbf{x}_i^{(k+1)} = \left\| \sum_{h=1}^H \sum_{j \in \mathcal{N}(i) \cup \{i\}} \alpha_{ij}^h \mathbf{W}^h \mathbf{e}_j^{(k)} \right\| \quad (4)$$

$$\alpha_{ij}^h = \frac{\exp(\text{LeakyReLU}(\mathbf{a}^{h\top} [\mathbf{W}^h \mathbf{e}_i \| \mathbf{W}^h \mathbf{e}_j]))}{\sum_{l \in \mathcal{N}(i) \cup \{i\}} \exp(\text{LeakyReLU}(\mathbf{a}^{h\top} [\mathbf{W}^h \mathbf{e}_i \| \mathbf{W}^h \mathbf{e}_l]))} \quad (5)$$

Node embeddings were refined using Multilayer Perceptrons (MLPs). This step included transformations, dropout for regularization, and non-linear activation functions, enriching the embeddings' expressiveness.

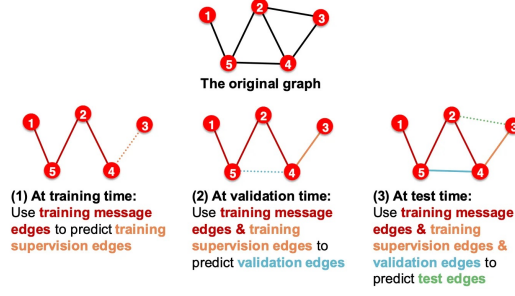


5.2 Splitting Improvement

We experimented with various splitting techniques to create balanced train, validation, and test sets while preserving the graph structure and label distribution. Initial attempts involved using RandomLinkSplit and custom stratified splitting logic, ensuring class proportions were maintained in each split. However, these approaches led to out-of-bounds errors and unsatisfactory results.

Unlike tabular data, where splitting is straightforward, graph data requires careful consideration of neighborhood structures. Randomly splitting nodes or edges risks disrupting these structures, which are critical for the model to learn meaningful relationships. To address this, edges were converted into Torch Geometric Data objects to ensure that splits preserved the graph's original connectivity and class proportions.

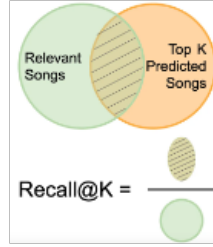
We finalized with splitting approach with a 70–15–15 percent split by keeping the true graph structure visible at all times but mask certain edges for prediction at each split. We call the visible edges “message passing” edges and the hidden ones we aim to predict “supervision” edges. This split results in the following behavior: During training, we use the train message passing edges to predict the train supervision edges. During validation, we use all train edges to predict the validation supervision edges. During testing, we use all train and validation edges to predict the test supervision edges.



6 Results

6.1 Evaluation Metrics

Recall@K For evaluation, Recall@K is a widely used metric, particularly in recommender systems focused on ranking. It calculates the proportion of relevant items that appear in the top K recommendations, helping gauge the model’s ability to surface items a user is likely to engage with. For example, in a music recommendation system, Recall@K might measure how many songs in a user’s playlist are correctly recommended among the top K items.



Bayesian Personalized Ranking (BPR) Loss Another key metric for training and evaluating recommendation models is the Bayesian Personalized Ranking (BPR) loss. BPR is designed to optimize the model’s ranking quality rather than just the accuracy of individual predictions. It works by comparing pairs of positive (relevant) and negative (irrelevant) items for each user, encouraging the model to rank positive items higher.

$$\text{BPR} = - \sum \log (\sigma(P_u^T Q_i - P_u^T Q_j))$$

The BPR loss function is computed as,

$$\text{BPR Loss}(i) = \frac{1}{|\mathcal{E}(i)|} \sum_{(i,j^+) \in \mathcal{E}(i)} \log \sigma(\text{score}(i, j^+) - \text{score}(i, j^-)) \quad (6)$$

where σ is the sigmoid function, $P_u^T Q_i$ is the score for a relevant element, $P_u^T Q_j$ and is the score for an irrelevant element. By minimizing this loss, the model learns to better differentiate relevant items from irrelevant ones, leading to more accurate and personalized recommendations.

6.2 Model Performance

The baseline achieved a recall of 0.4498, but the integration of GAT improved it to 0.5759. Adding MLP further boosted recall to 0.6961, with the lowest training and validation loss. The GAT mechanism allows GAT to weigh the importance of neighboring nodes, helping the model to emphasize the most relevant connections between playlists and tracks, while MLP’s non-linear learning enhanced embeddings.

Table 1: Performance comparison

	LightGCN - Gowalla	Baseline	Our Implementation	GAT	GAT + MLP
Recall	0.1830	0.4498	0.4710	0.5759	0.6961
Train Loss	-	0.5077	0.4843	0.2466	0.1622
Validation Loss	-	0.5182	0.4969	0.2851	0.2331

7 Conclusion

Complex Architectures Perform Better: Models with added complexity (e.g., GAT and GAT + MLP) outperform simpler approaches like LightGCN, both in recall and loss metrics. The attention mechanism and multi-layer perceptrons in GAT + MLP allow the model to focus on relevant graph relationships and refine embeddings for better predictions.

Trade-off Between Complexity and Performance: GAT-based models are computationally more expensive but result in significantly better performance metrics. Simpler models (e.g., LightGCN, Baseline) may be suitable for less complex tasks or resource-constrained environments.

7.1 Improvements and Future course of action

Based on the results and conclusions, here are some potential areas for future work and model improvements:

1. Enhanced Negative Sampling Techniques Observation: The current implementation includes random and hard negative sampling. Improvement: Explore adaptive negative sampling methods that dynamically balance between random and hard negatives based on model performance during training. Implement more sophisticated strategies, such as adversarial negative sampling, where negatives are selected to maximize model confusion.

2. Scalability for Larger Graphs Observation: Current methods may struggle with scalability due to computational cost, especially GAT-based models. Improvement: Use distributed training frameworks to scale the models for very large graphs (e.g., Spark or DGL). Apply graph sampling methods (e.g., GraphSAINT or Cluster-GCN) to work with subsets of the graph during training.

3. Incorporate Heterogeneous Graphs Observation: The current model simplifies the graph by excluding artist and album nodes. Improvement: Expand the graph to include heterogeneous node types (e.g., artists, albums) and edge types. Use heterogeneous GNNs (e.g., HGT or HAN) to capture richer relationships between entities.

4. Multi-Task Learning Observation: The current model focuses on playlist-track recommendations. Improvement: Introduce auxiliary tasks, such as artist or genre prediction, to improve the embeddings' quality. Use multi-task learning to optimize for multiple related objectives simultaneously.

5. Attention Mechanisms Observation: GAT uses attention to weigh node connections, but there may be room for improvement. Improvement: Explore more advanced attention mechanisms, such as multi-head attention with learnable weights or transformers, to enhance embedding quality. Investigate hierarchical attention to handle large graphs more efficiently.

6. Regularization and Optimization Observation: Overfitting was noted, especially with hard negative sampling. Improvement: Use dropout and weight decay more effectively to regularize the models. Employ advanced optimizers (e.g., AdamW or Lookahead) to stabilize training and convergence.

7. Incorporating Side Information Observation: The current model relies solely on playlist-track associations. Improvement: Integrate side information such as audio features, user metadata, or track metadata. Use multimodal embeddings to combine graph and feature-based information.

8. Real-Time Recommendations Observation: Models are designed for batch processing. Improvement: Adapt the models for online learning to handle dynamic graphs and real-time updates. Implement edge caching mechanisms to efficiently predict playlist-track links as new data arrives.

9. Interpretability Observation: GNNs are often considered black-box models. Improvement: Use explainability techniques (e.g., Grad-CAM or attention visualization) to interpret recommendations. Identify why certain tracks are recommended for specific playlists, enhancing user trust.

10. Alternative Loss Functions Observation: BPR loss is used for optimizing ranking-based objectives. Improvement: Experiment with other ranking-aware loss functions, such as pairwise hinge loss or listwise losses (e.g., ListMLE). Optimize for metrics like Recall@K directly using differentiable approximations.

Division of Work

Qianchi Huang – Explored and implemented the base model, ran ablations using GAT-MLP model
Shubham Singh – Explored splitting technique and experimented on dataset

References

- [1] Yin, T. (2023). Music Track Recommendation Using Deep-CNN and Mel Spectrograms. *Mobile Networks and Applications*, **28**(2130–2137). <https://doi.org/10.1007/s11036-023-02170-2>.
- [2] Maheshwari, C. (2023). Music recommendation on Spotify using deep learning. *arXiv preprint arXiv:2312.10079*. Retrieved from <https://arxiv.org/abs/2312.10079>.
- [3] Chen, C.W., Lamere, P., Schedl, M. & Zamani, H. (2018). Recsys Challenge 2018: Automatic Music Playlist Continuation. *Proceedings of the 12th ACM Conference on Recommender Systems (RecSys '18)*.
- [4] McFee, B. & Lanckriet, G. (2011). The Natural Language of Playlists. *Proceedings of the 12th International Society for Music Information Retrieval Conference (ISMIR '11)*, Miami, FL, USA, October.
- [5] Schedl, M., Zamani, H., Chen, C.-W., Deldjoo, Y. & Elahi, M. (2018). Current challenges and visions in music recommender systems research. *International Journal of Multimedia Information Retrieval*, **7**(2):95–116.
- [6] He, X., Deng, K., Wang, X., Li, Y., Zhang, Y. & Wang, M. (2020). LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation. *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*.
- [7] Eva, B., Thomas, B. & Benjamin, W. (2023, May 16). Spotify track neural recommender system. *Medium*. Retrieved from <https://medium.com/stanford-cs224w/spotify-track-neural-recommender-system-51d266e31e16>.