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. Our input is the Spotify Million Playlist Dataset, which we simplify by focusing on playlists and tracks as nodes in a graph. Edges connect playlists to their contained tracks, capturing associations. This structured representation provides the contextual and relational data needed to model playlist dynamics and generate song recommendations. Through this work, we propose an effective recommendation engine that contextualizes user preferences while utilizing advanced GNN-based architectures for improved recommendation accuracy. This approach highlights the potential for deep learning to further elevate personalized music experiences on streaming platforms.

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

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, more recent research by Chen et al. (2018) has found that the exact order of songs may not matter as much to users as previously assumed. Instead, the overall ensemble of songs and direct transitions between tracks seem to be more important for user satisfaction in playlist recommendations. Schedl et al. have further explored the task, identifying 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 deep CNNs alongside mel spectrograms to categorize music tracks and enhance music track recommendations. The model's ability to extract deep features from the audio signal enabled it to better capture user preferences, resulting in high-performance recommendations across various datasets, especially in contexts where traditional methods struggled. (Yin 2023) However, both Maheshwari and Yin's work focused on individual song features, which limits their ability to account for the collective structure within playlists. This is a crucial gap for automatic playlist continuation, as understanding how tracks interact and fit together within the context of a playlist is essential for more accurate recommendations.

3 Model Description

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. Below we describe each of the components in detail:

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. Negative Sampling Technique To obtain loss of the model, we need to define positive and negative edges for calculation. Positive edges are supervision edges we want to predict, while negative edges represent nonexistent connections paired with positive edges. Negative edges enforce the model to differentiate embeddings by encoding "Playlist i is like track j but not like track k. " For random negative sampling, negative edges are sampled randomly with a low probability of mistakenly picking a positive edge. For hard negative sampling, as the model starts training, it requires "easier" negatives so that the model can learn global neighborhoods. As the model starts to learn the features, edges that refine these neighborhoods to learn more local information are required. In order to obtain hard edges, a matrix of playlist-track scores is calculated, and sampled from top tracks.

3. Deep Learning Architectures The baseline model utilizes LightGCN, which 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.

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

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.

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

In our improvement, 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.

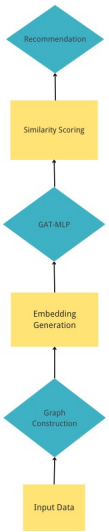
$$\mathbf{e}_i^{(k+1)} = \Theta \mathbf{x}_i^{(k+1)} + \mathbf{B} \tag{3}$$

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

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

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

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



(a) Model Structure

```
GCN                                      --
├─Embedding: 1-1                        2,259,200
├─ModuleList: 1-2                       --
│    └─Sequential: 2-1                  --
│    │    └─Linear: 3-1                 4,160
│    │    └─ReLU: 3-2                   --
│    │    └─Dropout: 3-3                --
│    │    └─Linear: 3-4                 4,160
│    └─Sequential: 2-2                  --
│    │    └─Linear: 3-5                 4,160
│    │    └─ReLU: 3-6                   --
│    │    └─Dropout: 3-7                --
│    │    └─Linear: 3-8                 4,160
│    └─Sequential: 2-3                  --
│    │    └─Linear: 3-9                 4,160
│    │    └─ReLU: 3-10                  --
│    │    └─Dropout: 3-11               --
│    │    └─Linear: 3-12                4,160
├─ModuleList: 1-3                       --
│    └─GATConv: 2-4                     960
│    │    └─SumAggregation: 3-13        --
│    │    └─Linear: 3-14                20,480
│    └─GATConv: 2-5                     960
│    │    └─SumAggregation: 3-15        --
│    │    └─Linear: 3-16                20,480
│    └─GATConv: 2-6                     960
│    │    └─SumAggregation: 3-17        --
│    │    └─Linear: 3-18                20,480
├─ModuleList: 1-4                       --
│    └─Linear: 2-7                      20,544
│    └─Linear: 2-8                      20,544
│    └─Linear: 2-9                      20,544
=================================================================
Total params: 2,410,112
Trainable params: 2,410,112
```

(b) Model Parameters

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, with track, artist, and album data. In total, the dataset has over 2 million unique songs from nearly 300,000 artists.

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.

In this project, the initial set-up 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 limited computational resources, we 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. 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.

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.

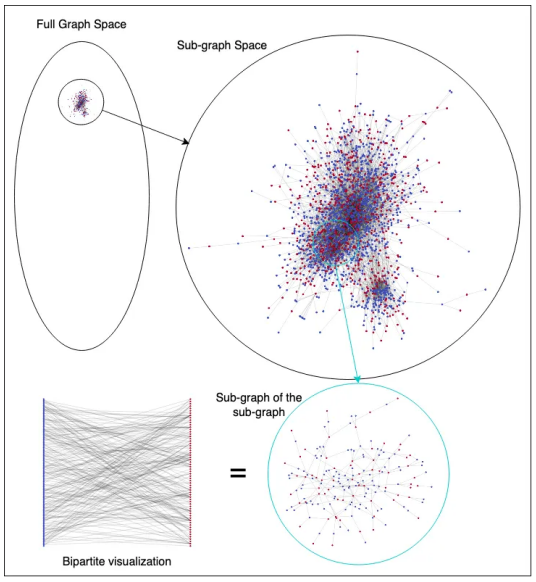


Figure 2: Subgraph Samples

5 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.

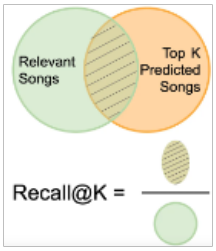


Figure 3: Recall@K

6 Loss Function

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.

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

The BPR loss function is computed as,

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

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.

7 Baseline Selection

The baseline model selected for this project is the LGCN based model integrating negative sampling from 2023. (Eva, B., Thomas, B., & Benjamin, W., 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.

8 Baseline Implementation

By adhering to the Light GCN model and other utilizations from torch geometric library and BPR Loss functions. The reproduction of the model's results aimed to closely match the performance metrics. We have implemented the model and trained it on the Spotify playlist dataset. The implementation loaded selected json files from the dataset and generated nodes and edges from the data and initiated the model with 4 LightGCN layers and BPR Loss.

Achieved Results The reproduced implementation achieved slightly better performance than the reported baseline results. Any minor deviations fall within expected variability for model reproductions due to factors such as data split randomness or small architectural tweaks that may occur in implementation.

We reached **Train loss** 0.4843; **Val loss** 0.4969; **Train ROC** 0.8253, **Val ROC** 0.8061, **Val recall** 0.4498.

9 Implemented Extensions

1. Integrating Graph Attention Networks (GAT) GAT enhanced the baseline LightGCN by introducing attention-based weighting, which allowed the model to dynamically assign varying importance to connections between nodes.

2. Embedding Refinement To improve the expressiveness of embeddings, we refined them using Multi-Layer Perceptrons (MLPs). MLPs introduce transformations to align embedding dimensions, incorporate dropout layers for regularization to prevent overfitting, and apply non-linear activation functions (e.g., ReLU) to add complexity to the feature space. Additionally, positional embeddings were added to track nodes to encode their position within a playlist. These embeddings provided spatial awareness, enabling the model to better represent the relationships between playlists and tracks.

3. Normalization: Normalization was applied during the recommendation generation process to stabilize the score outputs. By normalizing the similarity scores, the model ensures that track recommendations are consistent and comparable across different playlists. This step mitigates the impact of varying embedding magnitudes and enables the recommendation engine to focus on relative similarities, potentially improving the robustness of predictions.

4. Alpha as a Learnable Parameter: Experimented with alpha either as constant weight vector or learnable parameter across layers, where alpha can dynamically adjust layer-wise contributions during embedding propagation. By allowing the model to assign varying importance to different graph convolution layers, alpha optimized the aggregation of features across depths. This flexibility would improve the model's ability to adaptively balance local and global information for more accurate recommendations.

5. 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. (Stanford University, n.d.) 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.

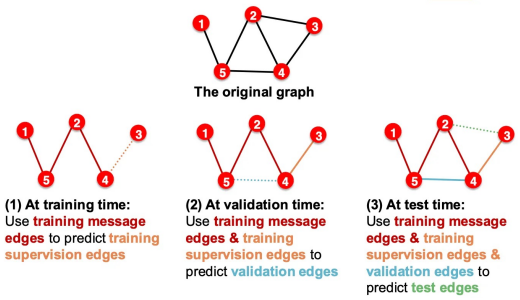
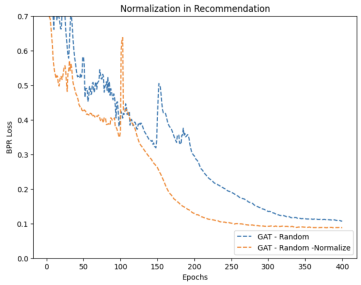


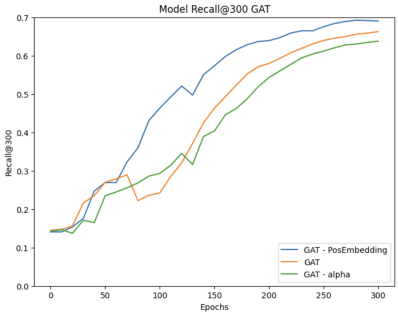
Figure 4: Transductive Link Prediction Split

10 Results

The baseline achieved a recall of 0.4498, but the integration of GAT improved it to 0.5759. Adding MLP further boosted recall to 0.6903, 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.

Aside from general model structure comparisons, we also experimented the integration of positional embeddings and the learnable alpha parameter, providing further improvements under specific conditions. When random negative sampling was used, the model managed to decrease loss but struggled to achieve a high recall. (See Model Recall Random) However, by incorporating positional embeddings and alpha, recall performance improved significantly, approaching the recall of 0.6931 comparable with hard negative sampling. Besides, normalization in recommending process also enhanced the model performance, where it significantly converged faster and obtained lower loss. When hard negative sampling was employed, the alpha parameter did not provide the same level of improvement. In this case, positional embeddings outperformed both alpha and random sampling methods, yielding lower loss and higher recall. (See Model Recall@300 GAT & BPR Loss of GAT) This indicates that positional embeddings were particularly beneficial when the model had access to high-quality negative samples, as they helped the model focus on the structure of the playlist.





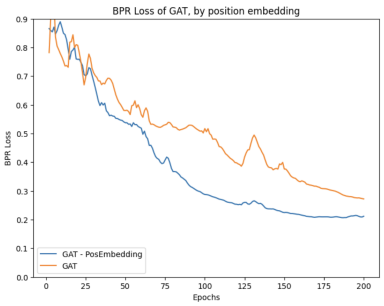


Table 1: Performance comparison

	LightGCN - Gowalla	Baseline	Our Implementation	GAT	GAT + MLP
Recall	0.1830	0.4498	0.4710	0.5759	0.6903
Train Loss	-	0.5077	0.4843	0.2466	0.1589
Validation Loss	-	0.5182	0.4969	0.2851	0.2384

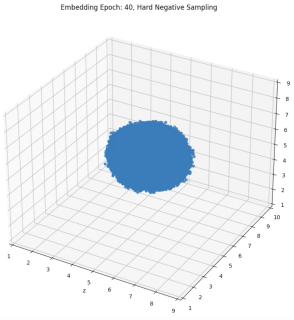
11 Discussion

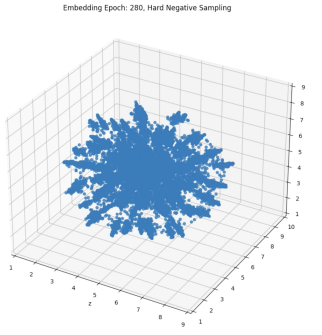
The experiment results display improved accuracy in recommendation performance. The integration of GAT allowed the model to better capture the importance of neighboring nodes, which significantly enhanced the recall. Positional embeddings proved useful in both hard and random negative sampling. In addition, when using random negative sampling, positional embeddings can help compensate for less informative input. On the other hand, learnable alpha did not contribute as expected under hard negative sampling, where positional embeddings alone achieved slightly better performance. This indicates that positional embeddings can better exploit the graph structure when high-quality negative samples are available, while alpha is more beneficial when the model faces less informative data.

The results also show that the model's sensitivity to the choice of combination of embedding and sampling strategy can significantly impact its performance. In cases where random sampling is used, incorporating positional embeddings and alpha helps enhance training by providing a richer representation of track positions within playlists. However, when high-quality negative samples are available, positional embeddings alone appear to be more effective in driving improvements in recall and reducing loss.

Further experiments could investigate the role of data sparsity and data noise. It would be interesting to explore how the model's performance changes when exposed to datasets with varying levels of noise or incomplete data. This would help validate the hypothesis that GAT-MLP and positional embeddings are especially beneficial when negative samples are less informative, as they might enhance learning even in challenging, sparse settings. Another avenue of exploration could involve testing models in real-world scenarios where user behavior might introduce unpredictable noise, to confirm whether the observed improvements hold up in more dynamic environments. In practical applications, the ability of a recommendation model to leverage graph structures and adapt to varying input structures could be crucial for generating accurate and personalized recommendations.

Aside from that, we would like to explore the propagation of embeddings in a visualized way. Here is the 3D representation using uniform manifold approximation and projection that reduced the original number of dimensions from 64 to 3. As the model starts training, the embedding shifts from forming a cluster and moving around the 3D space to gradually deviating from the initial cluster. As shown in the figure, at epoch 280, the embedding tended to form several brunches into clusters.





12 Future Works

The model achieved its stated objectives, but there are some potential areas for future work and model improvements.

The current implementation includes random and hard negative sampling. Improvements could be to explore adaptive negative sampling methods that dynamically balance between random and hard negatives based on model performance during training.

The current model simplifies the graph by excluding artist and album nodes. Improvement could be to 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.

Incorporating a hierarchical GAT can capture more multi-level relationships, and experimenting with different activation functions in the attention mechanism could also improve the flexibility and expressiveness of the attention scores.

Incorporating dynamism in playlist recommendation systems can be effectively achieved by leveraging Dynamic Graph Neural Networks (DGNNs), as outlined in "A Comprehensive Survey of Dynamic Graph Neural Networks." DGNNs integrate temporal features with graph-based learning to capture evolving relationships and patterns. Techniques like, Temporal Attention Mechanisms and Temporal Point Processes enable the model to assign dynamic weights to connections and predict future interactions. Utilizing frameworks like TGN or APAN, which efficiently handle continuous-time graphs, ensures real-time adaptability to user behaviour shifts. By employing time-aware embeddings and scalable memory mechanisms, DGNNs offer a robust approach to tracking and responding to temporal changes. We did not implement the dynamic component into our GNN but we have discussed it in our possible extensions within our team. Given the lack of research and resource constraints, we did not incorporate it.

Another similar line of thought as including dynamism in the model is by including time signature by assigning time-based weights to edges in the graph, reflecting the recency of interactions using the Spotify API.

13 Conclusion

The problem statement focused on addressing the challenge of personalized playlist recommendations on Spotify by utilizing graph-based methods. This project aimed 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.

Findings and Implications The integration of GAT and MLP significantly improved the model's ability to capture complex playlist-track relationships, resulting in higher recall (0.6961 compared to 0.1830 for LightGCN) and lower training/validation losses. These findings highlight the importance of dynamic edge weighting and enhanced feature learning, which have strong implications for real-world recommendation systems.

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.

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

Overall, the implementation achieved its original goal of improving playlist continuation, validating the effectiveness of graph neural networks combined with MLPs in this domain.

Division of Work

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

Code and Ablations

Please check out the code we use for our experiment here:

https://github.com/RBSand1/IDL-Project-Team-54

Our ablations here:

https://wandb.ai/qianchi55-carnegie-mellon-university/project-f24?nw=nwuserqianchi55

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.

[8] Feng, Z., Wang, R., & Wang, T. (2024, May 1). A comprehensive survey of dynamic graph neural networks: Models, frameworks, benchmarks, experiments, and challenges.Retrieved from https://arxiv.org/abs/2405.00476v1

[9] Stanford University. (n.d.). DeepSNAP: GraphDataset Module. Retrieved from https://snap.stanford.edu/deepsnap/modules/dataset.html#deepsnap-graphdataset

[10] Stanford University. (n.d.). CS224W: Machine Learning with Graphs. Retrieved from https://web.stanford.edu/class/cs224w/

[11] Rendle, S., Freudenthaler, C., Gantner, Z., & Schmidt-Thieme, L. (2009). BPR: Bayesian Personalized Ranking from Implicit Feedback. *Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence (UAI)*. Retrieved from https://arxiv.org/pdf/1205.2618

[12] Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., & Bengio, Y. (2017). Graph Attention Networks. *arXiv preprint arXiv:1710.10903*. Retrieved from https://arxiv.org/abs/1710.10903

[13] Neumeier, M., Tollkühn, A., Dorn, S., Botsch, M., & Utschick, W. (2023). Gradient Derivation for Learnable Parameters in Graph Attention Networks. *arXiv preprint arXiv:2304.10939*. Retrieved from https://arxiv.org/abs/2304.10939

[14] Aman. (n.d.). Recommendation Systems • Evaluation Metrics and Loss Functions. Retrieved from https://aman.ai/recsys/metrics/