Enhancing ”A Scalable Framework for Automatic Playlist Continuation on Music Streaming Services” with Album Covers

Tomer Laor

March 30, 2024

Abstract

In today’s world, music is in every aspect of our lives, facilitated by the widespread availability of streaming services. Playlists, integral to these platforms, empower users to curate thematic or genre-specific collections, with recommendations tailored to their preferences. This study delves into the realm of playlist continuation, enhancing the user experience by incorporating album covers as a means of augmenting recommendation frameworks on music streaming services. In this study, we use an existing scalable framework for automatic playlist continuation (APC) and enhance it by adding information from the vision modality to the recommendations, using embedding for album covers from pre-trained CLIP or DINOv2 models. In this work we improve the average clicks that a user has to make to get to the correct song in the recommendations list. We show that even though we also input images into the model and add more layers, the framework is still scalable and still acceptable for music streaming services.
Our code is accessible on GitHub [1].

1 Introduction

In an era dominated by digital music consumption, streaming services have revolutionized the way we access and experience music. Central to these platforms are playlists, which serve as curated collections tailored to users’ preferences, moods, and occasions. However, while playlists offer a personalized listening experience, the process of playlist creation and continuation remains a complex challenge, particularly in the context of automatic playlist continuation (APC).

APC poses a multifaceted challenge, given the diverse array of factors influencing song representation. A single track can be examined through multiple lenses, including its audio features, artist details, record label, release year, accompanying lyrics, user interactions, and more. These features can be broadly categorized into three distinct groups: metadata, encapsulating artist names, release dates, and related information; content, comprising the song’s audio and lyrical elements; and interactions, which encompass the engagement of various users with the song or engagement of users with the song’s artist or with the song’s album etc. As a task, APC is similar to information retrieval tasks [10], since the songs from the database should be ranked given a query (which is the current playlist). However, album covers present a unique challenge in categorization. Unlike metadata, they do not directly relate to the song’s intrinsic properties, yet they also do not fit neatly into the content category nor into the interactions category. This ambiguity prompts an exploration of leveraging album covers to augment existing methods. Moreover, [12] states that album covers represent the music contained inside them. [4] shows a correlation between the visual attributes of album covers such as colors, fonts, font sizes, number of written lines etc. to both genre and the artist’s origin country. These findings shows the value of album cover imagery as a rich source of relevant data.

Several existing methods rely solely on a single group of features. For instance, [13] exclusively utilize metadata, while [11] focus solely on content-based approaches. Some methods adopt a more comprehensive approach by leveraging a combination of feature groups. For example, [2] integrate metadata with playlist interactions (whether a song with this metadata value continued similar playlists or not), recognizing the significance of multiple dimensions in enhancing recommendation systems. To the best of our knowledge, our work is the first to incorporate album covers in an APC setting. By

[1] https://github.com/TomerMe2/APC-RTA-With-Album-Covers

incorporating a visual element, we aim to extend the representation power of songs and lead to better predictions.

In their work, [2] introduce a scalable framework for APC, named ”Represent-Then-Aggregate” (RTA). Within RTA, song embeddings are generated offline and aggregated online within the context of playlists to identify tracks with a high likelihood of playlist continuation. One of the key strengths of RTA lies in its extensible architecture. Our approach builds upon RTA by integrating embeddings of album covers, obtained from robust visual embedders such as CLIP or DINOv2, into the representation step. This addition enriches the feature set considered during playlist continuation. Furthermore, we adapt the aggregation step to accommodate these visual embeddings. While the original RTA framework leverages metadata and user interactions, our method extends this approach by incorporating metadata, user interactions, and album covers.

In Section 2, we'll delve into the fundamentals and review prior research on APC and image embedders. Additionally, we'll provide an overview of the RTA framework. In Section 3, we'll explain our methodology for assembling the album covers dataset and integrating it into the RTA framework. Furthermore, we'll present a straightforward example illustrating the application of visual embedders on album covers. Section 4 will outline our assessment strategy and the metrics employed. In Section 5, we'll present our discoveries, followed by an examination of their implications in Section 6, and conclude with final reflections in Section 7.

2 Background and Related Works

2.1 Automatic Playlist Continuation

In recent years, APC have gained attention due to the rising popularity of music streaming services such as Spotify, Apple Music, Youtube Music, Deezer etc. To push the academic front, Spotify released the Spotify Million Playlist Dataset (MPD) Challenge [3] in 2018. This dataset contains 1,000,000 playlists, created by users on the Spotify platform between January 2010 and October 2017. The evaluation task of this challenge is predicting the next song of a given playlist.

Interactions. Playlist continuation data is filled with interactions: user-song interactions, user-playlist interactions, song-playlist interactions, artist-playlist interactions, artist origin country-playlist interactions etc. Not all the datasets contains all the information required for all the interactions. For example, the MPD dataset doesn't contain information related to users thus user based interactions cannot be computed. [2] learn songs-playlist, artist-playlist, album-playlist, song duration-playlist and song popularity-playlist. They learn the interactions by apply one-hot encoding to songs, artists, albums, quantized songs durations and quantized songs popularities and they learn a weights matrix for each interaction that transforms the one-hot encoding into an embedding.

Metadata. Metadata is the information about the song which doesn't include the song content. Metadata for example are song title, artist name, artist origin country, release year, etc. [2] use embeddings learned for each metadata category. For example, they learn an embedding for each artist name in their train set. These embeddings are learned using a playlist continuation loss. [13] propose an algorithm to alleviate the problem of cold-start in APC (playlists that were just created without any songs yet) using the new paylist name. First, they cluster the palylists in their train dataset based on their names, as given by the users. Second, they apply SVD to create a matrix of song-playlist clusters. Third, they get the neighbor playlist cluster for a given song. Finally, they apply track weighting (using song frequency for example) and pick the n songs with the highest weight.

Content. Some works use the content of the song (the audio itself or the lyrics) to create APC algorithms. [11] creates a radio with 50 songs using a seed artist based on raw audio features. They use Mel Frequency Ceptral Coefficients (MFCC) coefficients, Rhythm features, beats per minutes and Onset Rate (number of beginning of notes or soungs per seconds) as features. They use modified anisotropic diffusion maps to reduce the dimensions of their feature space and create an embedding per song. To generate a radio, they require a seed artist. First they find the K-Nearest artists to the seed artist. Second they apply K-means on the seed artist songs and pick one random song from each cluster, these are reffered to as seed songs. Third they find the K-Nearest songs for each seed song. Then they pick the 50 songs that appeared the most in the k-Nearest neighbors of the seed songs according to diversity constraints.

2.2 Represent Then Aggregate

The Represent-Than-Aggregate (RTA) framework, introduced by [2], addresses the need for scalable yet effective APC models tailored for large-scale applications. This framework comprises two fundamental components: the representation step and the aggregation step.

In the representation step, denoted by $\phi(song)$, the framework applies a function to each individual song's data, thus creating a database of these transformed outputs. This process yields embeddings that encapsulate key features of the song, a task efficiently carried out offline since songs remain static after their release. On the other hand, in the aggregation step, denoted by $g(playlist)$, the framework applies a function to the dynamic playlist data, which comprises a list of embeddings corresponding to the songs in the playlist. This online process computes a representation of the playlist as a whole by using the embeddings of individual songs.

The inference mechanism of the entire framework involves computing a matching score between a playlist and a song. This is accomplished by applying the aggregation function g to the embeddings of songs within the playlist, followed by a dot product with the embedding of the target song. Formally, the matching score is expressed as:

$$MatchingScore(playlist, song_{target}) = g(\phi(song_1), \phi(song_2), ..., \phi(song_k)) \cdot \phi(song_{target}) \tag{1}$$

RTA is designed to be trainable end-to-end, facilitating seamless optimization across its components. They train and evaluate their framework on the MPD dataset. Given a batch of playlists, the framework employs a telescopic approach to generate $k-1$ sub-playlists from each playlist, where k denotes the length of the original playlist, by iterating over the playlist and use the songs up to this index as a sub-playlist. This strategy yields additional positive examples. In the context of RTA, positive examples correspond to the subsequent song within each sub-playlist. Conversely, negative examples are randomly sampled from all songs in the dataset, excluding those present in the respective sub-playlists. To optimize the framework, log-sigmoid loss is computed independently for both positive and negative examples. The results of these computations are then aggregated to yield the overall loss function. The losses are be formally defined as:

$$L_{\text{pos}}(p) = -\sum_{i=1}^{l-1} \log\left(\frac{1}{1+e^{-f(p_{:i}, p_{s_{i+1}})}}\right) \tag{2}$$

Where $-f(p_{:i}, p_{s_{i+1}})$ is the matching score that the model gives for a sub-playlist with i songs, denoted as $p_{:i}$, and the next song in the playlist, denoted as $p_{s_{i+1}}$. l is the number of songs in a playlist.

$$L_{\text{neg}}(p) = -\sum_{i=1}^{l-1} \sum_{s^- \in S^-(p)} \log\left(1 - \frac{1}{1+e^{-f(p_{:i}, s^-)}}\right) \tag{3}$$

Where $S^-(p)$ is a set of songs sampled from the songs dataset such that they do not appear in playlist p. $-f(p_{:i}, s^-)$ is the matching score that the model gives for a sub-playlist with i songs, denoted as $p_{:i}$ with the negative song sampled from the songs dataset, denoted as s^-.
The final loss is $L_{total} = L_{pos} + L_{neg}$.

[2] explored various options for the $\phi(song)$ function, including Matrix Factorization (MF), Factorization Machines (FM), and attention-based neural networks (NN). Similarly, they evaluated several alternatives for the $g(playlist)$ function, such as averaging, convolutional neural networks, gated recurrent units neural networks, and transformer decoder neural networks. Their evaluation metrics were categorized into three groups:

1. Accuracy Oriented: Precision, Recall, Normalized Discounted Cumulative Gain (NDCG), R-Precision, Clicks.

2. Popularity Oriented: Coverage, Popularity Bias.

3. Scalability Oriented: Training Time, Inference Time.

Among the evaluated models, those incorporating the transformer decoder as the aggregation method demonstrated superior performance. Interestingly, each representation method excelled in different metrics. For instance, Factorization Machines-based representations showed higher precision but lower Clicks compared to attention-based neural network representations.

2.3 Image Embedders

Deep neural networks are built using layers. The output of each layer contains information that should help the next layer. We will refer to the output of the last layer as an embedding. Embedding encapsulates the information that was discovered as the most helpful during training to fulfill the task that the neural network was trained to complete. In image classification neural networks, such as AlexNet [7], the embedding contains the most useful information that the neural network discovered during training to classify the image into the different classes in the dataset.

Self-supervision tasks are using the data itself, without annotations to learn good representations for the data. There was a big advancement in self-supervision models in recent years. An important milestone is Masked Autoencoders (MAE) [5], which trained an autoencoder architecture by masking the original image and predict the masked region. They show that by training a logistic regression using the embeddings from the decoder of their autoencoder as its data (also known as linear probing), they get 75.1% accuracy on ImageNet-1K. DINOv2 [8] is a model that was trained using self-supervision using images only as well, but it was trained to capture the same semantics of different views of an image, which results in an accuracy of 86.5% using linear probing on ImageNet-1K and 83.5 using KNN on ImageNet-1K. They show that MAE doesn't work well using KNN. DINOv2 success with KNN shows that it captures the semantics of the image so well, that it doesn't require a learning algorithm on its embedding to succeed in classification tasks.

Another way to train powerful image embedders is using text supervision, such as in CLIP [9]. CLIP is a model that was trained with pairs of images their text descriptions. The model was trained using contrastive learning to make the embeddings of the images and their descriptions as close as possible while making the embeddings of the images and other descriptions in the batch as farther away as possible in the embedding space. Their distance function is dot product. They show that in a zero-shot setting (wihout seeing any of the images from this dataset), just by applying 1-NN with the embeddings of the text of the labels of ImageNet-1K, they get 76.2% accuracy on ImageNet-1K using CLIP.

3 Method

3.1 Dataset Creation

We used the MPD dataset. MPD contains 1,000,000 playlists, 2,262,292 unique tracks and 734,684 unique albums [3]. MPD is distributed with Spotify's album URI of each track and is not distributed with the album cover images. Spotify WebAPI [1] has an endpoint that accepts up to 20 albums URIs and returns the link for the album covers for each album. We used this endpoint to get the links to the album covers of each album. Some albums had no links, and some link were broken. After getting the link to the album cover, we downloaded it and applied pad and rescale to achieve a $224x224$ image and saved the transformed image. We chose this size since this is the input shape of CLIP and DINOv2. We saved the image after the transformation to save disk space and allow faster loading times. We applied padding to keep the original aspect ratio. The pad color was gray. After collecting all the images, we passed all the images through CLIP and DINOv2 to acheive their embeddings and we save these embeddings to the disk of our server. We ended up with 733,149 album covers embeddings, which are 99.8% of the albums in MPD.

Challenges. Spotify WebAPI has rate limits, which described vaguely as of this day. They write that Spotify's API rate limit is calculated based on the number of calls that your app makes to Spotify in a rolling 30 second window, but they don't write how many calls in 30 second window will lead to exceed the rate limits. Spotify WebAPI blocked us after a certain amount of time (usually around 3 hours) even when keeping a consistent amount of requests per 30 seconds. This challenge made us creating 5 different WebAPI credentials, since each block is for around 24 hours. We were able to finish the collection of the album covers after several days in which we changed the Spotify WebAPI credentials after each block.

3.2 Enhance Represent Then Aggregate Using Album Covers

Pipeline overview. RTA [2] is built from 2 learnable components: Representor and Aggregator. The representor inputs information about a song and outputs an embedding of this song. [2] used one-hot

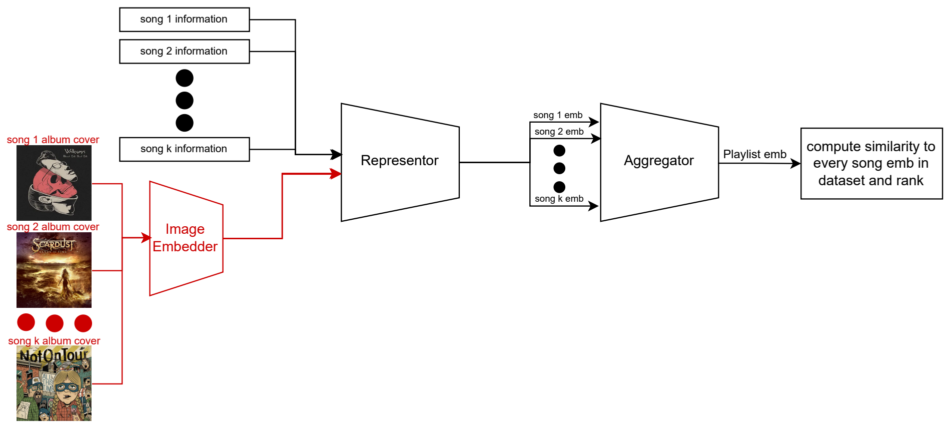


Figure 1: Overview of our method. Our additions to RTA are highlighted in red. Our addition is feeding the album cover of each song to an image embedder (CLIP or DINOv2), and feed those embeddings to the representor alongside the information about the song. Each song is fed into the representor seperately, and the list of embeddings that the representor creates (one for each song) is fed into the aggregator.

encodings of artist, album, quantized duration and quantized popularity. They quantize the duration and the popularity to make a categorical representation from continues values. In Figure 1 we can see the overview of our method. We add information about the album covers to the representor by creating an embedding for the album cover using an image embedder (CLIP [9] or DINOv2 [8] in this work), and feed it into the representor alongside the song information, as used in [2]. Computing the similarity to songs in the dataset remains the same as in [2]. We decided to fuze the information of album covers in an early stage of the model (known as early fusion). Most methods in music recommendations that uses multiple inputs use late fusion [10], which means that they use an aggregate function on the output of multiple predictors to get a single prediction. We have a single predictor with a single output thanks to incorporating early fusion.

Representor. Figure 2 shows the architecture of our modified representor to handle the introduction of the album cover embedding to the representor input. The modification that we make to the representor is using a song information representor that can be Matrix Factorization (MF), Factorization Machines (FM) or Neural Network decoder (NN), as described in [2] which accepts the song information, and using a fully connected (FC) neural network that will serve as our adaptor between the album cover embedding space and the song embedding space. We then concat these embeddings to create the final song embedding. The adaptor is a neural network composed from a FC layer followed by ReLU followed by FC layer. The adaptor is important since we wish to transform the embeddings that we get from the pre-trained image embedders into embeddings that will be learned to fit in the song embedding correctly. Moreover, the score between the playlist embedding and the song embedding to know which song to recommend is computed using dot product. Dot product assumes that each element in the vector has equal importance. Therefore, album cover embedding with a big length relative to the song information embedding will skew the dot product computation to use mostly the information about the image and it will give less weight to the song information. This leads us to make the embedding that the FC adaptor outputs to be proportional in size to the embedding of the song information. The embedding size of CLIP is 512 while the embedding size of DINOv2 is 768, which is way too big compared to the size of the song information embedding which is 128. We pick the embedding size that the FC adaptor outputs to be 64.

Aggregator. The second learnable component is the Aggregator. The aggregator accepts a list of song embeddings and creates a playlist embedding. We didn't change the aggregator's architecture, but the input dimension of the aggregator has changed since our song embedding is bigger because we concat the embedding of the song information representor with the embedding of the image embedder.

Missing Album Covers. Some album covers will be missing. They can be missing from various reasons: broken links, corrupted files, etc. 99.8% of the album in MPD are present in our album covers dataset. If a song's album cover is missing, we will create an array of 0s as the output of the image embedder and will continue the pipeline normally.

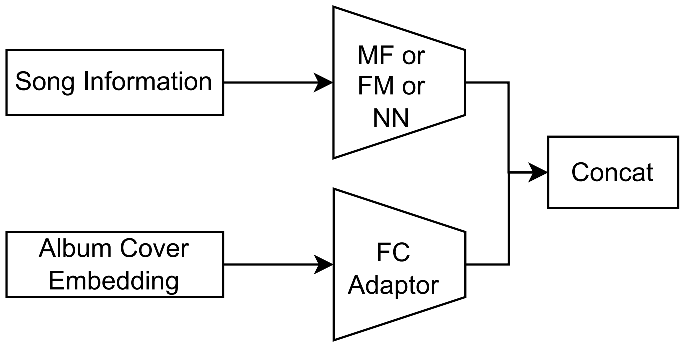


Figure 2: Our proposed change for the representor. We support MF, FM and NN from the original RTA. The song information is fed into the MF, FM or NN as in the original representor, and the album cover embedding from CLIP or DINOv2 is fed into a fully connected (FC) neural network which lowers the dimensionality of the album cover embedding. The output of the representor is achieved by concat the embeddings after the MF, FM or NN and the FC adaptor

3.3 Anecdotal Proof of Concept

Here, we aim to provide an anecdotal proof of concept, demonstrating a potential connection between album covers and the respective artists. This exploration holds significance as previous studies, such as [2], leverage artist information for recommendations. By establishing a link between album covers and artists, we highlight that album covers contain relevant information utilized in existing recommendation methods.

To conduct our experiment, we collected albums from the top 10 most popular artists in the MPD dataset. Subsequently, we obtained CLIP and DINOv2 embeddings for each album and visualized them using t-distributed Stochastic Neighbor Embedding (t-SNE) to create 2D scatter plots.

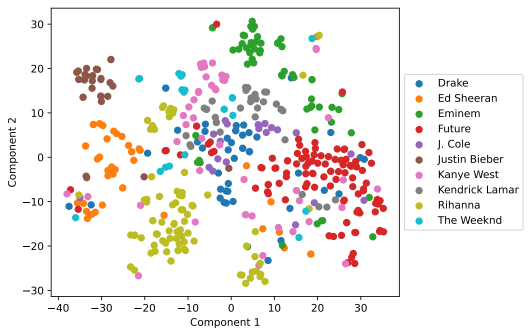
In Figure 3, we observe that both CLIP and DINOv2 embeddings exhibit clusters representing albums from the same artist. Remarkably, these embeddings were not generated with explicit artist information, and the t-SNE model was not trained with artist data.

Notably, the CLIP embeddings demonstrate superior clustering of album covers to their respective artists in this particular instance. Noteworthy patterns emerge, such as Justin Bieber's album covers being closer to Ed Sheeran's in the CLIP scatter plot. Furthermore, an intriguing observation is the formation of a diagonal line from the top left to the top right corner of the CLIP 2D embeddings space. The bottom-left triangle predominantly contains albums of pop artists, while the top-right triangle primarily features albums of rap artists. The Weekend's genre is R&B, which resides somewhere between pop and rap, a characteristic also reflected in this scatter plot.

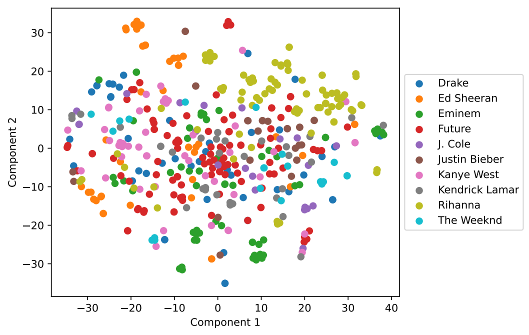
4 Evaluation

4.1 Experimental Plan

The models that show the best performance in [2] are those that their aggregator is the Transformer. We trained our method using all base represntors from [2] (MF, FM, NN) with the Transformer aggregator. To train the models, we took the parameters that [2] found to be the best using hyper-parameter optimization and we applied them on the models that are enhanced with album covers as well. There are 18 parameters that are used for each training. The set of parameters are listed in the resources directory in our GitHub repository. We didn't apply hyper-parameter optimization on our method. We ran all of our experiments with such that the last layer in the FC adaptor in our representor has a size of 64 neurons.



(a)



(b)

Figure 3: 2D Scatter Plots of CLIP (a) and DINOv2 (b) Embeddings After t-SNE Dimensionality Reduction.

4.2 Metrics

We evaluated our method using the metrics used in [2]. They used metrics defined in [3] and added more metrics.

Accuracy Oriented Metrics:

- Normalized Discounted Cumulative Gain (NDCG): a measure for ranking quality. A metric that was devised for information retrival tasks [6]. It increases when the ground truth songs are placed higher in the ranked list of recommended candidates.

- R-Precision: this metric is based on precision. The difference is that a recommended song that is not the ground truth song, but from the same artist as the ground truth, gets a score of 0.25 instead of 0 when computing the precision [3]. Formally, R-Precision is defined as:

$$R - Precision = \frac{|S_T \cap G_T| + 0.25 \cdot |S_A \cap G_A|}{|G_T|} \tag{4}$$

  Where G_T is the set of ground truths, S_T is the set of the recommended track IDs, G_T is the set of unique track IDs in the ground truth, S_A is the set of artist IDs of the recommended songs and G_A is the set of unique artist IDs in the ground truth.

- Clicks: Spotify have a feature that recommends 10 songs in every batch. This metric counts how much clicks the user will have to make to reach the song that is the ground truth song. The maximum number of clicks is 50 (determined by Spotify), so when a song is not in the first 500 recommendations, this playlist continuation gets a clicks count of 51.

Popularity Oriented Metrics: In an APC we would like a to have a balance between exploration and exploitation [10]. We would like to be able to recommend non popular songs where it fits. These metrics shows us how much popularity bias our APC system has.

- Coverage: the percentage of songs from the catalog recommended at least once during the test phase.

- Popularity: shows the popularity of the recommended songs. Computed by counting the occurrences of each song in the training set and applying min-max normalization. Low scores indicates less popular songs are recommended.

Scalability Oriented Metrics: [2] used training time as a measure for scalability. We had a problem measuring training time since we trained using a GPU cluster that has multiple models of GPUs and CPUs, which results in different training times. For this reason we omit the computation of the training time. All of our trainings completed in less than 25 hours. A metric that we will compute is inference time, which is the average time required to recommend a list of 500 candidates for a song [2].

Models	Precision (%)	Recall (%)	R-Precision (%)	NDCG (%)	Clicks	Popularity (%)	Coverage (%)
MF-Transformer as reported in [2]	5.20 $\pm$ 0.09	39.76 $\pm$ 0.47	22.30 $\pm$ 0.28	29.04 $\pm$ 0.38	2.60 $\pm$ 0.17	10.46 $\pm$ 0.13	5.35
MF-Transformer reproduced	**5.21 $\pm$ 0.09**	**39.78 $\pm$ 0.47**	**22.34 $\pm$ 0.28**	**29.09 $\pm$ 0.38**	2.57 $\pm$ 0.17	10.63 $\pm$ 0.13	5.20
CLIP-MF-Transformer	5.07 $\pm$ 0.09	38.96 $\pm$ 0.47	21.79 $\pm$ 0.27	28.33 $\pm$ 0.37	**2.48 $\pm$ 0.16**	**9.63 $\pm$ 0.12**	**6.49**
DINOv2-MF-Transformer	5.1 $\pm$ 0.09	39.15 $\pm$ 0.47	22.01 $\pm$ 0.27	28.62 $\pm$ 0.37	2.59 $\pm$ 0.17	9.72 $\pm$ 0.12	5.61
FM-Transformer as reported in [2]	5.31 $\pm$ 0.09	40.46 $\pm$ 0.47	22.29 $\pm$ 0.27	29.21 $\pm$ 0.37	2.52 $\pm$ 0.17	**11.74 $\pm$ 0.13**	10.18
FM-Transformer reproduced	**5.34 $\pm$ 0.09**	**40.7 $\pm$ 0.47**	**22.42 $\pm$ 0.27**	**29.33 $\pm$ 0.37**	2.42 $\pm$ 0.16	12.03 $\pm$ 0.13	10.24
CLIP-FM-Transformer	5.28 $\pm$ 0.09	40.2 $\pm$ 0.47	22.06 $\pm$ 0.27	28.86 $\pm$ 0.37	**2.32 $\pm$ 0.15**	11.77 $\pm$ 0.13	10.64
DINOv2-FM-Transformer	5.27 $\pm$ 0.09	40.15 $\pm$ 0.47	22.03 $\pm$ 0.27	28.85 $\pm$ 0.36	2.48 $\pm$ 0.16	11.89 $\pm$ 0.13	**10.98**
NN-Transformer as reported in [2]	**5.26 $\pm$ 0.09**	**40.17 $\pm$ 0.47**	21.18 $\pm$ 0.27	**29.14 $\pm$ 0.37**	**2.17 $\pm$ 0.15**	10.33 $\pm$ 0.13	11.81
NN-Transformer reproduced	5.11 $\pm$ 0.09	39.13 $\pm$ 0.47	21.54 $\pm$ 0.27	28.3 $\pm$ 0.37	2.34 $\pm$ 0.15	9.34 $\pm$ 0.13	**16.29**
CLIP-NN-Transformer	5.09 $\pm$ 0.09	39.01 $\pm$ 0.47	21.52 $\pm$ 0.27	28.27 $\pm$ 0.37	2.39 $\pm$ 0.15	**9.26 $\pm$ 0.12**	15.85
DINOv2-NN-Transformer	5.13 $\pm$ 0.09	39.33 $\pm$ 0.47	**21.74 $\pm$ 0.27**	28.54 $\pm$ 0.37	2.24 $\pm$ 0.15	9.64 $\pm$ 0.13	16.06

Table 1: Performance comparison of models, all models have an inference time of below 0.01 seconds for a playlist.

5 Results

We conducted experiments using Transformer aggregators exclusively, employing the MF, FM, and NN representors from [2]. For each representor, we present metrics from the original study [2], our reproduction of their model's performance, and our method's performance using CLIP and DINOv2 embeddings.

We successfully replicated the results of MF-Transformer and FM-Transformer, but encountered challenges with NN-Transformer that prevented us from achieving results within the confidence intervals reported by [2] although running the code published by [2].

All of our models exhibits an inference time of less than 0.01 seconds per playlist, which makes them acceptable for music streaming services according to [2].

For MF-Transformer, incorporating album cover embeddings from CLIP reduced the average number of clicks required for users to reach the correct song, from 2.6 to 2.48. However, using DINOv2 embeddings did not reduce the number of clicks. Additionally, album cover embeddings led to a significant decrease in popularity, from 10.46% to 9.63% (CLIP) and 9.72% (DINOv2). Nevertheless, there was an increase in coverage, from 5.25% to 6.49% (CLIP) and 5.61% (DINOv2). Other accuracy-oriented metrics were negatively impacted.

For FM-Transformer, we observed an improvement in the clicks metric, decreasing from 2.52 to 2.32 (CLIP, statistically significant improvement) and 2.48 (DINOv2). Similarly, coverage increased from 10.18% to 10.64% (CLIP) and 10.98% (DINOv2), while other accuracy-oriented metrics and popularity were negatively impacted.

For NN-Transformer, there was a statistically significant improvement in R-Precision, rising from 2.18% to 21.52% (CLIP) and 21.74% (DINOv2). Popularity also saw a substantial increase, from 10.33% to 9.26% (CLIP) and 9.64% (DINOv2). Additionally, reported coverage improved from 11.81% to 15.85% (CLIP) and 16.06% (DINOv2), although our reproduction for NN-Transformer yielded a coverage of 16.29%. However, other metrics suffered as a result of our method.

The representor utilized in our evaluations is illustrated in Figure 2. Attempts to augment this representor with an additional fully connected layer after the concat function to combine album cover features with song data did not yield favorable results. For instance, the model with an additional fully connected layer for CLIP-FM-Transformer produced R-Precision of 20.2%, NDCG of 26.6%, and Clicks of 3.17, all worse than any other method in our evaluation.

6 Discussion

In this study, we introduced a novel approach to augment the Represent-Than-Aggregate (RTA) framework by incorporating album cover information using frozen pre-trained image embedders, specifically CLIP or DINOv2. Our findings demonstrate notable improvements in reducing the number of clicks required for users to reach the ground truth song in the MF-Transformer and FM-Transformer methods when employing the CLIP visual embedder.

However, we encountered challenges in reproducing the training of the NN-Transformer using the code provided in the original paper, which consequently undermines our confidence in the integration of our method with the NN-Transformer. Despite this limitation, we observed a reduction in the number of clicks for the NN-Transformer that we trained, albeit unsuccessfully, particularly with the DINOv2-NN-Transformer.

Furthermore, our method exhibited the potential to decrease the popularity of recommended songs and enhance the coverage of recommendations across the entire song database. However, it is worth noting that in most cases, our approach badly affected other accuracy-oriented metrics such as Precision, Recall, R-Precision, and NDCG.

Among the accuracy oriented metrics, we consider Clicks to be the most pertinent in evaluating the user experience of playlist continuation recommendation systems, especially within the context of Spotify, where the graphical user interface (GUI) displays only 10 recommendations at a time, as described in [3].

Due to the computational resources required, each training run consumed approximately 25 hours on Nvidia RTX 3090 GPUs. Due to the significant computational demands, we were unable to conduct hyper-parameter optimization. Consequently, we relied on the hyper-parameters identified by [2] for each method, which may have contributed to suboptimal results.

7 Conclusion

In this study, we curated a dataset comprising 733,149 album covers, encompassing 99.8% of the albums in the MPD dataset. Through visual analysis using CLIP and DINOv2 embeddings of the top 10 artists from the MPD dataset, we observed clear separations among artists and even genres, underscoring the potential of album covers as informative features.

Our key contribution lies in enhancing the RTA framework for automatic playlist continuation. We propose integrating album cover information into the algorithm by augmenting the representor with CLIP or DINOv2 embeddings. This involves adding fully connected layers to the representor and concatenating these embeddings with the song-level information.

Our method exhibits an inference time of less than 0.01 seconds, meeting the criteria set forth by [2] for music streaming services. Notably, our modification reduces user interaction, as evidenced by fewer clicks required to reach the desired song in Spotify's interface.

Moreover, our approach enhances coverage for algorithms that we were able to reproduce, expanding the portion of the song database recommended by our system. Our method also privdes a decrease in the popularity of recommended songs for the MF representor. These findings underscore the efficacy of integrating album cover information into the RTA framework for enhancing playlist continuation algorithms.

References

[1] Spotify webapi. https://developer.spotify.com/documentation/web-api. Accessed: 2024-03-23.

[2] Bendada, W., Salha-Galvan, G., Bouabça, T., and Cazenave, T. A scalable framework for automatic playlist continuation on music streaming services. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval* (2023), pp. 464–474.

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

[4] Dorochowicz, A., and Kostek, B. Relationship between album cover design and music genres. In *2019 Signal Processing: Algorithms, Architectures, Arrangements, and Applications (SPA)* (2019), IEEE, pp. 93–98.

[5] He, K., Chen, X., Xie, S., Li, Y., Dollár, P., and Girshick, R. Masked autoencoders are scalable vision learners. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (2022), pp. 16000–16009.

[6] Järvelin, K., and Kekäläinen, J. Cumulated gain-based evaluation of ir techniques. *ACM Transactions on Information Systems (TOIS) 20*, 4 (2002), 422–446.

[7] Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems 25* (2012).

[8] Oquab, M., Darcet, T., Moutakanni, T., Vo, H., Szafraniec, M., Khalidov, V., Fernandez, P., Haziza, D., Massa, F., El-Nouby, A., Assran, M., Ballas, N., Galuba, W., Howes, R., Huang, P.-Y., Li, S.-W., Misra, I., Rabbat, M., Sharma, V., Synnaeve, G., Xu, H., Jegou, H., Mairal, J., Labatut, P., Joulin, A., and Bojanowski, P. Dinov2: Learning robust visual features without supervision, 2024.

[9] Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., Krueger, G., and Sutskever, I. Learning transferable visual models from natural language supervision, 2021.

[10] Schedl, M., Knees, P., McFee, B., and Bogdanov, D. Music recommendation systems: Techniques, use cases, and challenges. In *Recommender Systems Handbook*. Springer, 2021, pp. 927–971.

[11] Shuhendler, R., and Rabin, N. Dynamic artist-based embeddings with application to playlist generation. *Engineering Applications of Artificial Intelligence 129* (2024), 107604.

[12] Vad, M. The album cover. *Journal of Popular Music Studies 33*, 3 (2021), 11–15.

[13] Yürekli, A., Kaleli, C., and Bilge, A. Alleviating the cold-start playlist continuation in music recommendation using latent semantic indexing. *International Journal of Multimedia Information Retrieval 10*, 3 (2021), 185–198.